- IPMI -

IPMI CIM Mapping Guideline

Document Revision 0.60
2/6/2006

Intel Hewlett-Packard Dell Avocent

Intellectual Property Agreement and Disclaimers

The information contained in this IPMI CIM Mapping Guideline ("Mapping Guideline") represents the current view of Intel Corporation ("Intel"), Dell Incorporated ("Dell"), Avocent Corporation ("Avocent") and Hewlett Packard Company ("Hewlett Packard") on issues discussed as of the date of publication. Because Intel, Dell, Avocent and Hewlett Packard must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Intel, Dell, Avocent and Hewlett Packard, and Intel, Dell, Avocent and Hewlett Packard cannot guarantee the accuracy of any information presented. This Mapping Guideline is for informational purposes only.

Intel, Dell, Avocent and Hewlett Packard (collectively referred to herein as the "Authors") provide this Mapping Guideline under the following licensing terms. This Agreement to Royalty Free Licensing Terms for IPMI CIM Mapping Guidelines Agreement and the corresponding Intellectual Property Agreement and Disclaimers for the Mapping Guideline makes reference to the IPMI CIM Mapping Guideline. This Agreement to Royalty Free Licensing Terms for IPMI CIM Mapping Guidelines Agreement and the corresponding Intellectual Property Agreement and Disclaimers does not replace or supersede any of the Promoter, Contributor, or Adopter agreements, Intellectual Property agreements, or disclaimers for any IPMI specifications, including, but not limited to, the agreements for the Intelligent Platform Management Interface Specifications and the Intelligent Platform Management Interface Second Generation Specifications.

THE MAPPING GUIDELINE IS PROVIDED "AS-IS" AND THE AUTHORS DO NOT MAKE ANY REPRESENTATION OR WARRANTY REGARDING THE MAPPING GUIDELINE OR ANY PRODUCT OR ITEM DEVELOPED BASED ON THE MAPPING GUIDELINE. THE AUTHORS DISCLAIM ALL EXPRESS AND IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND FREEDOM FROM INFRINGEMENT. WITHOUT LIMITING THE GENERALITY OF THE FOREGOING, THE AUTHORS DO NOT MAKE ANY WARRANTY OF ANY KIND THAT ANY ITEM DEVELOPED BASED ON THE MAPPING GUIDELINE, OR ANY PORTION OF THEREOF, WILL NOT INFRINGE ANY COPYRIGHT, PATENT, TRADE SECRET, OR OTHER INTELLECTUAL PROPERTY RIGHT OF ANY PERSON OR ENTITY IN ANY COUNTRY EXCEPT AS PROVIDED FOR BELOW. IT IS YOUR RESPONSIBILITY TO SEEK LICENSES FOR SUCH INTELLECTUAL PROPERTY RIGHTS WHERE APPROPRIATE.

THE AUTHORS SHALL NOT BE LIABLE FOR ANY DAMAGES ARISING OUT OF OR IN CONNECTION WITH THE USE OF THE MAPPING GUIDELINE, INCLUDING LIABILITY FOR LOST PROFIT, BUSINESS INTERRUPTION, OR ANY OTHER DAMAGES WHATSOEVER.

Mapping Guideline Definitions. "Necessary Claims" shall mean those claims of all patents and patent applications throughout the world, excluding any claims to enabling technologies that may be necessary to make or use any implementation of the Mapping Guideline but are not themselves expressly set forth in the Mapping Guideline (e.g., semiconductor manufacturing technology, basic operating system technology, etc.), which a party has the right to grant licenses of the scope granted herein without such grant or the exercise of rights thereunder resulting in payment of royalties or other consideration to third parties (except payment to affiliates) and which are necessarily infringed by an implementation of this Mapping Guideline pursuant to the preceding and following sections of this Intellectual Property Agreement, where such infringement could not have been avoided by another technically reasonable non-infringing implementation of the Mapping Guideline.

Grants of Licenses. Each Author hereby grants, to each of the other Authors and all implementers of the Mapping Guideline, a nonexclusive, perpetual, royalty-free, nontransferable, nonsublicenseable (except as part of transfer of an end user product), worldwide license under its Necessary Claims to make, have made, use, import, offer to sell and sell and otherwise distribute the portions of products, whether hardware, software, or combination of hardware and software, which implement all or part of the Mapping Guideline for the sole purpose of mapping the IPMI 1.5 Specification into a DMTF CIM specification. No license is granted hereunder to any person or entity to implement either IPMI or DMTF specifications.

Authors' grant of the above license to each implementer is subject to the implementer's agreement to grant, upon request, a nonexclusive, perpetual, royalty-free, nontransferable, nonsublicenseable (except as part of transfer of an end user product), worldwide license to the Authors, under such implementer's Necessary Claims to make, have made, use, import, offer to sell and sell and otherwise distribute the portions of products, whether hardware, software, or combination of hardware and software, which implement all or part of the same version of the Mapping Guideline for the sole purpose of mapping the IPMI 1.5 Specification into a DMTF CIM specification.

Document Feedback and Contributions. Any feedback or other contribution by any person or entity in any country that may be incorporated into this document can only be provided if the contributor agrees in writing to the IPMI CIM Mapping Feedback Agreement. The IPMI CIM Mapping Feedback Agreement can be obtained by contacting one of the Authors.

Representation and warranties. Each Author represents and warrants that to the actual knowledge of the Author's named representative (1) the Author has full rights to make the contributions made by the Author to the Mapping Guideline, (2) no other party has provided feedback or contributed to the Mapping Guideline through the Author and that any party doing so in the future must have agreed to do so as a Contributing Party by executing the IPMI CIM Mapping Feedback, and (3) the exercise of the licenses granted hereunder by the Author with regard to the Author's contributions will not, to the knowledge of the Author, violate any third party copyright or trade secrets.

Copyright. The Authors hereby grant a nonexclusive, perpetual, irrevocable, non-transferable, worldwide royalty free copyright license, to make copies, distribute, publish and display the Mapping Guideline so long as the below copyright notice is retained with each copy of the Mapping Guideline.

This document is not for sale.

To obtain additional copies of this this document, please download the source files from the web sites at http://www.intel.com/design/servers/ipmi

Abstract

The Common Information Model (CIM)[1] is an object-oriented information model defined by the Distributed Management Task Force (DMTF), which provides a conceptual framework for describing management data.

This document defines CIM mappings for specific areas of server management. This includes the behavioral conventions required for interoperability. This is intended to be a consistent, cross-vendor mapping of the Intelligent Platform Management Interface (IPMI)[2] to CIM.

Referenced Standards

Cross-references to standards being published or developed by DMTF (or other organizations), or reproduction of information from published standards, are for the purpose of interoperability with those standards and for defining functionality needed for interoperability. Such cross-references are not for the purpose of using copyrighted material from those standards. All such referenced content remains outside of the scope of this document. See Reference Documents for additional information.

Change History

Version	Date	Editor	Description
0.60	February 06, 2006	Jon R. Hass, Dell Inc.	Added the following major sections: - IPMI Entity Mapping - IPMI Watchdog Mapping - IPMI Software Identity Mapping Updated Section 2 – Common Steps for Mapping Process. Changes to Sensor property mappings. Changes to Log property mappings. Changes to User ID Management mappings. Changes to BMC mappings. Changes to Versioning mapping behavior.

Open Issues

1 Common Information Model (CIM) Specification, V2.2, June 14, 1999, http://www.dmtf.org/standards/standard_cim.php

2 Intelligent Platform Management Interface Specification, V1.5, February 20, 2002, http://www.intel.com/design/servers/ipmi

92 # Outline Table of Contents

109

110 # Table of Contents

511

512 # List of Figures

534

List of Tables

1 Introduction

Note on providing feedback: Feedback or comments on this IPMI CIM Mapping Guideline should be sent to: jon_hass@Dell.com, tom.slaight@intel.com, steve.lyle@hp.com, and steffen.hulegaard@avocent.com. Any feedback provided must be done on a royalty free licensing basis. For feedback to be considered and incorporated into this IPMI CIM Mapping Guideline, the submitter must agree in writing to the IPMI CIM Mapping Feedback Agreement. See the directions in the Copyright/IP Disclaimer above.

The purpose of the IPMI-CIM Mapping Guideline is to ensure interoperability in the usage of CIM classes for (primarily) server management. Additionally, the IPMI-CIM Mapping Guideline defines a consistent way to connect service processors (SPs) that implement IPMI compliant control interfaces to CIM based control interfaces. Such service processors (SPs) are often called baseboard management controllers (BMCs).

1.1 Target Audience

The IPMI CIM Mapping Guideline is intended for use by developers of IPMI-to-CIM translation/mapping logic.

1.2 Conventions

The key phrases and words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as in RFC 2119.

1.3 Reference Documents

[1] Common Information Model (CIM) Specification, V2.2, June 14, 1999 - Downloadable from http://www.dmtf.org/spec/cims.html

[2] Unified Modeling Language (UML) from the Open Management Group (OMG) - Downloadable from http://www.omg.org/uml/>

[3] Intelligent Platform Management Interface (IPMI) Specification, February 20, 2002 - Downloadable from http://www.intel.com/design/servers/ipmi/

[IPMI 1.5] Refers to the *Intelligent Platform Management Interface Specification v1.5, revision 1.1*, February 20, 2002. Copyright © 2002 Intel Corporation, NEC Corporation, Dell Computer Corporation, and Hewlett-Packard Company, plus the corresponding errata and addenda document [IPMI Errata], referenced below.

[IPMI 2.0] Refers to the combination of two documents: *Intelligent Platform Management Interface Specification - Second Generation, v2.0*, revision 1.0, February 2, 2004. Copyright © 2004 Intel Corporation, Hewlett-Packard Company, NEC Corporation, Dell Computer Corporation. Plus: [IPMI Errata] Available from http://www.intel.com/design/servers/ipmi

602 [IPMI Errata] Intelligent Platform Management Interface Second Generation Specification,
603 v2.0, revision 1.0, Intelligent Platform Management Interface Specification v1.5,
604 revision 1.1, Addendum Document, Revision 1, June 1, 2004. Copyright © 2004
605 Intel Corporation, Hewlett-Packard Company, NEC Corporation, Dell Computer
606 Corporation. Available from http://www.intel.com/design/servers/ipmi

607 [CIM 2.9] DMTF CIM 2.9 Schema, Copyright © 2005 Distributed Management Task Force,
608 Inc. (DMTF). All rights reserved.

609 [CIM 2.10] DMTF CIM 2.10 Schema, Copyright © 2005 Distributed Management Task
610 Force, Inc. (DMTF). All rights reserved.

611

612 1.4 Terminology

613

Term	Definition
in-band	Dependent upon either hardware and/or software resources used for supporting the principal workload.
in-service	Dependent on software, services and/or interfaces that require a concurrently running operating system.
out-of-band	Independent of both hardware and/or software resources used for supporting the principal workload.
out-of-service	Independent of software, services and/or interfaces that require a concurrently running operating system.
SM-Bus	System Management Bus

614

615 1.5 Acronyms and Abbreviations

616

Acronym	Definition
BMC	Baseboard Management Controller
BT	Block Transfer (in-band IPMI interface)
CLP	Command Line Protocol (from the DMTF WBEM/CIM SMASH initiative)
DMTF	Distributed Management Task Force (www.dmtf.org)
I2C	Inter-Integrated-Circuit bus (a 2/3 wire bus for certain in-band IPMI interfaces)
IB	In-band (see Terminology section)
ICMB	Intelligent Chassis Management Bus (part of IPMI)
IPMB	Intelligent Platform Management Bus (part of IPMI)
IPMI	Intelligent Platform Management Interface (www.intel.org/design/servers/ipmi)
KCS	Keyboard Controller Style (in-band IPMI interface)
LAN	Local Area Network
MAP	Manageability Access Point
OOB	Out-of-Band (see Terminology section)
PCI	Peripheral Component Interconnect (Bus - http://www.webopedia.com/TERM/P/PCI.html)
SDR	Sensor Data Repository (defined by IPMI)
SDRs	Sensor Data Records (which are resident in the IPMI SDR)
SMASH	Systems Management Architecture for Server Hardware
SMIC	System Management Interface Chip (Interface – an in-band IPMI interface)
SP	Service Processor
SSIF	SM-BUS System Interface (an in-band IPMI interface)
WBEM	Web Based Enterprise Management (Initiative within the DMTF)

1.6 Editorial Conventions

By default, we'll use IPMI terms in each property/method mapping section. If you see a term like 'SensorType', this means the IPMI SensorType (not the CIM SensorType). If we need to refer to a CIM property/method, we will prefix with the corresponding CIM class name and a 'dot' separator. Here is an example:

CIM_Sensor.SensorType

Sometimes, a CIM property/method reference will be scoped to a specific specialization rather than CIM base class that introduces a particular property and/or method. What follows is an example of such a specialization scoped reference

CIM_NumericSensor.SensorType

- **Editorial Shorthand**

Notation	Meaning
:=	"Defined as" or "Defined to be"
=>	Maps to

1.7 Notational Conventions

- **Operators**

Operator	Example	Description
<>	<SDR Type 1 bytes 49+> <CIM_Sensor.Name>	These brackets are a substitution macro. The <…> bracket enclosed text references something whose corresponding value should be substituted. For example, Table 37-1 in the IPMI v1.5 specification describes bytes 49+ in the Type 1 IPMI Sensor Data Record (SDR) as being the sensor's ID string bytes. Reuse and the ***Define Once Discipline***: The <CIM_Sensor.Name> example references the elsewhere defined mapping for the CIM_Sensor class's Name property. In the <CIM_Sensor.Name> example, the previously defined mapping is being referenced and re-used (rather than defined in more than one place). We especially encourage such reuse when references cross documents and/or chapter divisions.
(<type>)	(Boolean) (String)	Cast/convert to the given fundamental type. For example, a binary value [0,1] might be casted/converted to a CIM boolean value ["true","false"]. We often use the cast operator to emphasize the types of the values involved. These redundant/unnecessary casts (of say a string value to a string value) are simply of way of making the mapping specification more self-documenting.
EStr()	EStr(SensorType)	Maps a value to an associated description string. This is the "explain" string macro/function. It amounts to mapping an enumeration's value to an associated enumeration's symbolic value. This also covers any other similar sort of associative array based mapping-to-string. For example, IPMI v1.5 Sensor Type (hex defined) numeric value is defined in Table 36-3. Each value is associated in Table 36-3 with a corresponding description string.
TruncAt(<integer>, string)	TruncAt(5, "Hereisabigstring") == "Herei"	Truncates a sequence at a specified number of entries. Often used to truncate a string (a sequence of characters) to a given length. If the string is shorter than the given length, TruncAt indicates no change to the input string (i.e. a no-op).
Relops (=,>,<,etc.) LogicalOps (&&, ||)	SensorType=02h	Relational operators represent filtering/scoping predicates. For example, SensorType=[01h-04h] means SensorType values limited to the range 01h to 04h. Notice that this mixes the equivalence operator and a regular expression. The same example could be denoted as follows: SensorType>=01h && SensorType<=04h Or SensorType>00h && SensorType<05h

||	"foo" || "bar" => "foobar"	String concatenation. The '||' operator is a logical 'OR' if it's operands are relational expressions. It is string concatentation otherwise (and it requires string type operand context to function as string concatenation).

- **Regular expressions**

  Some common regular expression syntax may be intermingled in the mapping definitions.

  If we eventually need both a rigorously defined and an advanced regular expression syntax, our direction is to conform the Python "re" module regular expression syntax (see Programming Python, 2nd Edition, Lutz, Mark, O'Reilly & Associates, Inc, Sebastopol, California, 2001, ISBN 0-596-00085, p1041+). The Python regular expression syntax deliberately subsumes the Perl regular expression syntax.

- **Python style pseudo-code (when and if necessary)**

  Since Python was designed to be a C/C++ style pseudo-code, we'll adopt the Python language conventions for any procedural mapping specifications. We do not intend to write executable Python code. We do not imply anything about Python nor Python's suitability for any given purpose. Our interest in Python is only as a means to express any necessary IPMI-to-CIM procedural mapping logic (fragments). We adopt Python's widely familiar, C/C++ style, syntax . This is just an expedient choice of procedural specification languages.

  When, and if, we need to resort to procedural mapping specifications, we'll introduce the corresponding Python fragments with a "*Procedure*" heading. The Python-like pseudo-code will indulge in some non-Python notation where the possibility for confusion is low and where the notation reinforces connections to either IPMI terminology/notation and/or CIM terminology/notation. For example, IPMI constant notation like "03h" might be used rather than Python's "0x3". We also use compact self-assignment forms from C/C++ (like ||= for concentration assignment). This is just more concise and often more convenient.

- In-line comments

  In-line (pseudo-code) comment syntax follows the Python conventions.

  # This is a comment until end of current line. Multi-line comments

  # much use a '#' character to introduce each comment line.

1.8 Referenced Standards

Standard/Specification	Version	Source	Citation
CIM Infrastructure Specification	2.3	DMTF	DSP 0004, www.dmtf.org/standards/cim
CIM Schema	2.10	DMTF	www.dmtf.org/standards/cim

Standard/Specification	Version	Source	Citation
CIM Specification	2.2.2	DMTF	DSP 0007, www.dmtf.org/standards/cim
IPMI Specification	1.5	Intel	www.intel.org/design/servers/ipmi
IPMI Specification	2.0	Intel	www.intel.org/design/servers/ipmi

664

2 Common Steps for Mapping Process

This section presents the general process that instrumentation provider software would use for accessing an IPMI-based platform management subsystem, discovering its available management capabilities, and mapping those capabilities to CIM objects described in this document. These steps are presented as a guide to software design and to applying the IPMI specifications. These steps are summarized as follows:

1. Discover and connect to the IPMI interface for the particular managed system.
2. Discover and enumerate the IPMI Entities, Sensors, and FRU information creating a logical 'list' of IPMI capabilities.
3. Take this logical listing and use it to instantiate and populate CIM objects and associations according to the discovered capabilities on the given system.

2.1 Discovering IPMI Interfaces

Instrumentation that maps from IPMI to CIM must first discover and connect to the IPMI platform management subsystem via one of the IPMI interfaces. The IPMI interfaces may be classified into three types: System, Remote, and Internal.

2.1.1 System Interfaces

The System Interface is a set of registers that software running on the managed system uses to directly access the IPMI subsystem. There are three types of system interface defined in [IPMI 1.5]: the KCS (keyboard controller style) interface, the BT (block transfer) interface, and the SMIC (server management interface chip) interface. Of these, the KCS interface is by far the most widely deployed, followed by the BT interface. The SMIC interface is an older interface that is not recommended for new designs. [IPMI 2.0] introduced a fourth system interface, called SSIF (SMBus System Interface) that is used with low cost IPMI BMCs (baseboard management controllers).

Local instrumentation software must decide which types of System Interface it will support. It is highly recommended that the KCS interface be supported. Other interfaces should be considered based on the target systems for the software. The IPMI Web Site, http://www.intel.com/design/servers/ipmi, has reference drivers and sample code that can assist with the development of software for a number of the system interfaces.

Section 6 of the IPMI specifications describes the System Interfaces and their operation. In this section, there is a subsection titled "System Interface Discovery and Multiple Interfaces". This subsection describes the steps that software should use to discover the type of system interface used on a given platform, its location, and how to handle the situation if multiple system interfaces are present.

2.1.2 Remote Interfaces

The IPMI platform management subsystem on a given system may also be accessible remotely via IPMI remote interfaces. The remote interfaces include serial/modem and LAN (Ethernet).

The IPMI remote interfaces incorporate their own media-specific protocols for delivering IPMI messages to the platform management subsystem. However, the IPMI messages themselves are the same regardless of the interface over which they're delivered. Thus, in general, the same IPMI management capabilities that can be accessed locally can also be accessed via the remote interfaces[3]. This means that the same general process of mapping IPMI capabilities to CIM locally (via the system interface) can also be used for mapping IPMI to CIM via the remote interfaces. This enables the possibility of creating 'proxy' software to provide a CIM-based interface to remote IPMI-based systems.

Because the interfaces provide remote access, the serial/modem and LAN interfaces incorporate options for securing the interfaces and restricting access to specific users that are have been configured and enabled for IPMI access to a particular BMC. Thus, a remote application must not only support the protocol requirements for deliver IPMI messages over the interface, but also include support for 'logging in' to the remote system as a user that has the necessary privilege level to support IPMI-to-CIM mapping.

Unless otherwise indicated, the full set of capabilities enabled by IPMI-to-CIM mapping can be accomplished by a user that has "Operator" privilege. Refer to Appendix G in the IPMI specifications for a listing of the privilege levels for the IPMI commands.

IPMI over Serial/Modem has several different access protocols that can be used. These are referred to as "Basic Mode", "Terminal Mode", and "PPP Mode". Basic Mode support is mandatory for IPMI over Serial/Modem, the other modes are optional. An application can therefore rely on Basic Mode as being common across platforms that support IPMI over Serial/Modem.

The approach to handling user logins varies between the serial access protocols. However, all authenticated interfaces share the same association of user names and privilege levels. The PPP interface also incorporates the option for an additional layer of "link" security where the serial communication connection itself must first be authenticated, and then IPMI users are authenticated on top of that. An application that supports IPMI over PPP should also support the options for link-level security.

IPMI over LAN includes support for enabling applications to discover IPMI-based systems on the network. The serial/modem interfaces presume that the application has a-priori knowledge of the phone number necessary to connect to the managed system, or that there is a direct connection available between the managing (console) system and the IPMI serial interface. However, once the serial connection is established, the interfaces include mechanisms that allow the availability of IPMI to be confirmed.

It should be noted that the serial interfaces may also be used with a capability called "Serial Port Sharing" where access to the system's serial controller can be shared with IPMI access. This guideline does not cover CIM-based instrumentation access to the system serial connection via IPMI Serial Port Sharing. A remote application should be aware, however, that it will need to take additional steps to switch the serial connection to the BMC if Serial Port Sharing is in effect.

[3] Typically, only certain interface-specific IPMI commands will not be available on other interfaces. For example, commands related to LAN discovery and session setup are not available over other interfaces.

2.2 Internal Interfaces

IPMI also defines interfaces that can be used to enable other management controllers, add-in cards, and other devices to access IPMI 'inside the box'. These internal interfaces include the IPMB (Intelligent Platform Management Bus), the PCI SMBus, and an inter-chassis bus referred to as the ICMB[4] (Intelligent Chassis Management Bus). (While the ICMB bus extends between chassis, it is more characteristic of an internal interface because it is considered to be a 'physically secure' and therefore does not include 'user logins' for controlling user access.

As with the remote interfaces, the same IPMI management capabilities that can be accessed via the System Interface can also be accessed via the internal interfaces. This means that the same general process of mapping IPMI capabilities to CIM can be used, creating the possibility of an add-in card that incorporates the IPMI-to-CIM mapping to provide a CIM-based remote interface for the managed system.

2.3 Discovery and Enumeration of IPMI Entity, Sensor, and FRU Information and Associations

Once software is able to access IPMI, the next major step of the mapping process is for software to discover and enumerate the particular platform's IPMI capabilities so they can be mapped into CIM. This includes discovering the control capabilities, sensors, FRU (field replaceable unit) information, and the associations between sensors and FRU information for the entities that are monitored and managed through IPMI.

This discovery and enumeration process can be thought of as creating a logical 'list' of the IPMI capabilities for a given platform. Once this logical listing is created, it can be used to drive the instantiation of CIM objects for accessing those capabilities.

2.3.1 Recommendations on the access and use of SDR Info

The IPMI Sensor Data Records (SDRs) are a key element to the enumeration and discovery of IPMI sensors, FRU information, and the entities they are associated with. In most cases, it will be faster and more efficient for software to read out the entire set of sensor data records in bulk into a local file or system RAM and work with the copy of the SDRs rather than to access SDRs and process them one at a time.

In addition, if an application is responsible for monitoring multiple systems, it is useful to cache information about the managed systems rather than accessing it anew each time the managed system is accessed. IPMI provides timestamps on the SDR Repository (accessed using the *Get SEL Info* command) that software can use to tell if the SDR contents have changed since the last time the info was accessed.

[4] There is a separate document that describes the ICMB and its operation. Refer to the reference to [ICMB] in the IPMI specifications.

2.4 Entity Discovery

IPMI supports a type of SDR called "Entity Association Records". These records allow physical or logical Entities to grouped together under a common 'container' Entity. The container Entity may itself be a 'contained' Entity in another Entity Association. Thus, Entity Associations can form a 'tree' hierarchy expressing relationships between Entities and/or groups of Entities.

There can be an interdependency between IPMI Sensors and Entities. If the container Entity at the top of a given Entity Association or Entity Association Hierarchy is absent (as indicated by a Presence sensor or by presence bit associated with a sensor for the Entity) then all 'contained' Entities under that hierarchy are considered to be absent. Consequently, any Sensors associated with those contained Entities are also considered absent.

Because multiple sensors can be associated with a single Entity, and because the presence of multiple Entities can be determined by a the presence of an Entity Association or Entity Association Hierarchy, it is generally more efficient to determine the presence of Entities first, then Sensors.

Whether an Entity Association is 'present', or 'absent' is based on whether the container Entity for the Association has sensors associated with it, and if so, what the state of those sensors is, as follows:

- If the container Entity does not have any sensors associated with it, then the association is assumed to be 'present'.
- If the container Entity has one or more sensors associated with it, but all sensors are 'scanning disabled' or are inaccessible, then the association should be considered 'absent' and should be ignored. I.e. software should act as if that association was not even listed in the SDRs.
- If the container Entity has one or more 'scanning enabled' associated with it, and none of those sensors are a presence sensor or presence bit that indicates that the container Entity is absent, then the association is assumed to be 'present' (valid).
- If the container Entity has a presence sensor or bit associated with it that indicates the container Entity is absent, then the overall association is said to be '*explicitly absent*' and the container Entity and any contained Entities and lower level associations should not be instantiated because of being listed under that association. (Note that the same Entities may still become instantiated because they're listed under a different Entity association or association hierarchy that *is* present.)

2.4.1 Entity presence rules

The following is an brief overview of the different ways Entity presence can be determined in IPMI. This is described in more detail in 10.3.2 Entity Instantiation Algorithm. In this section, the words "the Entity" shall be used as a shorthand for "The particular instance and type of Entity".

2.4.1.1 Explicit by Entity Association

The Entity listed under an "Entity Association" that is defined by one or more Entity Association Records and is present

AND

None of the Entity Associations that are higher up in the Entity hierarchy are explicitly absent.

AND

The Entity is not listed in the Entity ID/Entity Instance fields of the SDR for a Presence sensor, nor a sensor that includes a presence bit that indicates the Entity is 'not present' or 'not installed' (see note below).

2.4.1.2 Explicit by Sensor SDR or Event SDR

The Entity is not listed under *any* present or explicitly absent Entity Association Hierarchy, but is listed in the Entity ID/Entity Instance fields of one or more "sensor" or "event only" SDRs (Type 01h, 02h, or 03h).

AND

The Entity is not listed in the Entity ID/Entity Instance fields of the SDR for a Presence sensor, nor a sensor that includes a presence bit that indicates the Entity is 'not present' or 'not installed'(see note below).

AND

At least one sensor associated with then Entity is enabled (See 2.5, Sensor Discovery, for the definition of an 'enabled' sensor).

2.4.1.3 Explicit by FRU Device Locator Record

The Entity is not listed under *any* Entity Association Hierarchy (regardless of whether the hierarchy is present or absent), but is listed in the Entity ID/Entity Instance fields of one or more FRU Device Locator SDRs (Type 11h).

AND

There are no sensors associated with the Entity, or at least one sensor associated with then Entity is enabled (See 2.5, Sensor Discovery, for the definition of an 'enabled' sensor).

AND

The Entity is not listed in the Entity ID/Entity Instance fields of the SDR for a Presence sensor, nor a sensor that includes a presence bit that indicates the Entity is 'not present' or 'not installed' (see note below).

AND

The FRU Device for associated with the Entity is accessible by software.

2.4.1.4 Implicit by Sensor Type

The existence of an Entity can also be inferred by the Sensor Type. For example, if a Fan sensor is present, then it implies the existence of a Fan. This is done only when the Sensor Type and the Entity ID do not match, and only for certain Sensor Types, listed in a table in Section 10.3.1, Mapping Entity ID's to CIM Classes. An Entity that has its presence inferred this way is not explicitly identified by IPMI Entity ID/Entity Instance numbers. Therefore, for instance identification purposes, some unique ID must be synthesized. In this specification, a combination of the Sensor Number, Owner ID, and other information, is used to synthesize a CIM Sensor.DeviceID value that is used as the instance identifier.(see note below). Thus, Entity can be inferred to exist if:

The Entity the is explicitly associated with the sensor (the Entity identified by the Entity ID and Entity Instance fields in the SDR) is present by one of the two preceding explicit presence mechanisms.

AND

The Sensor Type in the Sensor Data Record is of a type that implies the existence of a physical Entity, per Table 16 "Physical" Sensor Types to Entity Mapping .

AND

The Entity ID implied by the Sensor Type in the Sensor Data Record for the sensor does not match the explicit type given by the Entity ID in the Sensor Data Record.

AND

At least one sensor associated with then Entity is enabled (See 2.5, Sensor Discovery, for the definition of an 'enabled' sensor).

2.4.1.5 Implicit by FRU Data

The existence of an Entity can also be inferred by IPMI FRU data. For example, if there is Chassis information in a FRU Device, it implies the existence of a physical chassis. This mechanism is used when there is not a FRU Device Locator record for the FRU Device that calls out a particular Entity ID and Entity Instance for an Entity to be associated with the FRU data. As with Entities whose existence is "Implicit by Sensor Type" it is necessary to synthesize an instance identifier in lieu of having explicit Entity ID and Entity Instance information.

FRU Information Type:

- Chassis Info Area, implies existence of a Chassis Entity.
- Board Info Area, implies existence of a board or module.

Refer to the steps outlined in Section 10, IPMI Entity Mapping, and Table 16 "Physical" Sensor Types to Entity Mapping.

Note:

The two presence mechanisms (Presence sensor or presence bit) should not be used simultaneously in an implementation. However, if an implementation does utilize both mechanisms with a given Entity, the following interpretation rule applies

- the Entity should be considered to be 'not present' (absent) if either mechanism reports 'not present'.
- the Entity should be considered to be 'present' if both mechanisms report present
- if scanning is disabled for the sensor for either explicit mechanism, the remaining mechanism is utilized as if the other didn't exist.
- if scanning is disabled for both mechanisms, the rules for the Entity Presence Sensor apply - meaning that ALL sensors associated with that Entity should be ignored.

Refer to the IPMI Specifications to find out which sensor types include sensor-specific offset bits that serve as presence bits for associated Entities.

2.5 Sensor Discovery

This step consists of the discovery of 'enabled' sensors or 'event only' sensors where the Entity associated with the sensor is part of an Entity Association or Entity Association Hierarchy that is present, or the Entity is not covered by an Entity Association or Entity Association Hierarchy.

An 'enabled' sensor is a sensor that responds to a *Get Sensor Reading, Get Sensor Event Enable,* or *Get Sensor Event Status* that returns bit [6] as 1b, indicating 'sensor scanning enabled'. An 'event only' sensor is a sensor that is listed using the 'type 03h' Event Only SDR.

The Entity that is associated with a sensor is identified by the Entity ID and Entity Instance fields in the SDR for the sensor.

If a sensor is not listed in the SDRs it should be ignored. I.e. should be considered 'non-existent' with respect to this Sensor Discovery step.

2.5.1 Sensor Presence Rules

Thus, as sensor should be considered to be present (and a candidate for CIM instantiaton) if:

The sensor is listed in the SDRs
AND
The sensor is 'scanning enabled'
AND

The Entity associated with the sensor is not part of an Entity Association or Entity Association Hierachy

OR

The sensor is listed in the SDRs
AND
The sensor is 'scanning enabled'
AND
The Entity associated with the sensor is part of an Entity Association or Entity Association Hierarchy that is present (refer to 2.4.1, Entity presence rules, above).

2.6 FRU Device Discovery

IPMI FRU Devices hold FRU (Field Replaceable Unit) information for different replaceable Entities in the system. This can include information such as serial number, model, manufacturer, part number, etc. FRU Devices are accessed by software either as 'logical' or 'physical' FRU Devices. Logical FRU Devices are accessed by sending *Read-* and *Write FRU Data* IPMI command to the management controller that provides access to the device.

Physical FRU Devices (also referred to as 'non-intelligent' FRU Devices) are accessed as separate SEEPROM (Serial Electrically Erasable Programmable Read-Only Memory) devices on a physical I^2C or SMBus segment. Thus, physical FRU devices are often used provide FRU information on add-in cards, since only a simple SEEPROM is required to provide the information, rather than a management controller that can handle IPMI commands. To access physical FRU devices, software uses the IPMI *Master Write-Read* command to generate SEEPROM access transactions on the target I^2C or SMBus behind a management controller.

Logical and Physical FRU Device access is described in more detail in the IPMI Specification. In particular, refer to the chapter 38, Accessing FRU Devices, and the documentation on the *Read-* and the *Master Write-Read* command.

FRU Devices are discovered by the following mechanisms:

2.6.1 By BMC Primary FRU Device

Since IPMI requires a BMC to be present, the BMC is not required to have a Management Controller Device Locator record in the Sensor Data Records. The BMC has a "Primary FRU Device" which is accessed as logical FRU Device 0 on LUN 00b of the BMC using the *Read-* and *Write FRU Data* IPMI commands.

2.6.2 By Management Controller Device Locator Record

Management Controllers other than the BMC are listed using individual Management Controller Device Locator (Type records in the Sensor Data Records. A "Device Capabilities" field in the record includes a bit that indicates whether or not the management controller supports a "Primary FRU Inventory Device". I.e. has FRU information that is accessible as logical FRU Device 0 on LUN 00b of the management controller using the *Read-* and *Write FRU Data* IPMI commands.

2.6.3 By FRU Device Locator Record

FRU Devices that are not Primary FRU Devices are listed using individual FRU Device Locator records in the Sensor Data Records. The record provides information on whether the device is accessed as a Physical or Logical FRU Device.\

2.7 Discovering Sensor, Entity, & FRU Associations

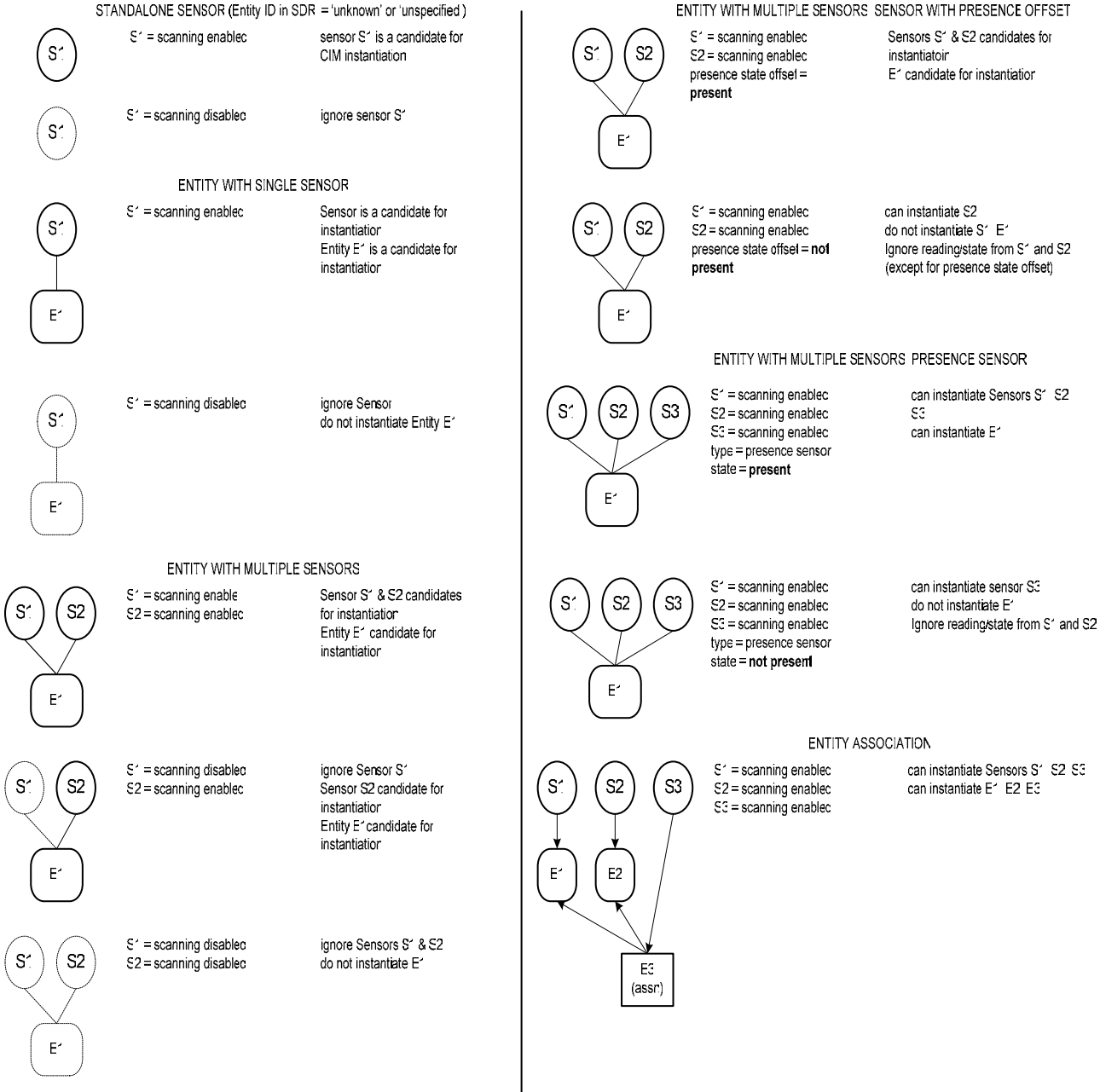
The following figures graphically represent some common inter-relationships between sensors and Entities and how they relate to sensor and entity presence. In general, software uses the following information sources to determine and generate CIM associations between sensors, Entities, and FRU information:

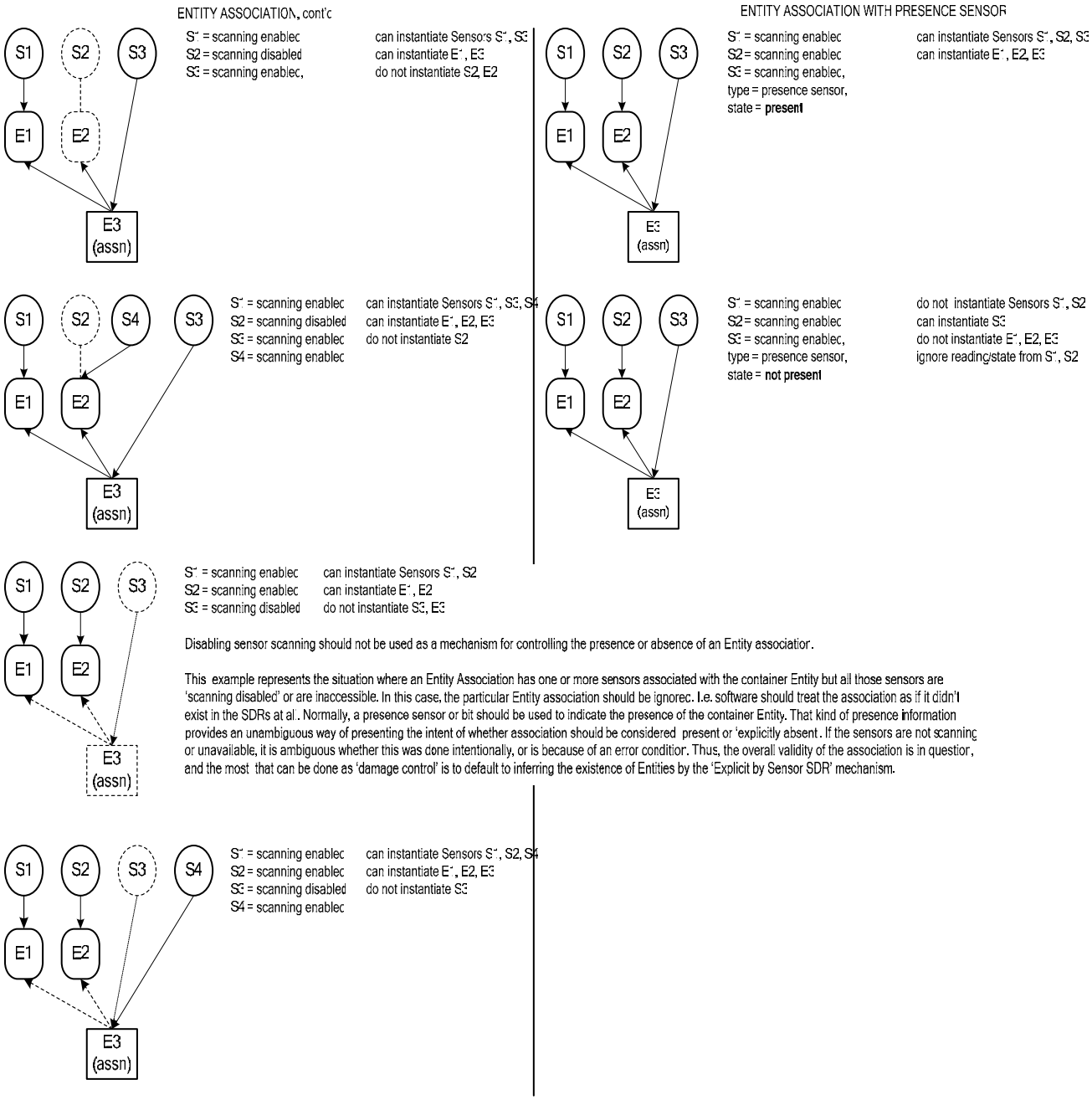
- **Entity ID Information in sensor SDRs:** Each sensor is associated with a single Entity, and there can be a multiple sensors associated with a given Entity. Software uses the Entity ID and Entity Instance information to identify the Entity that the sensor is associated with.
- **Entity Association records:** Entities can be associated with one another in a container/contained relationship via Entity Association records. Entity Association Records can be nested to form an "Entity Association Hierarchy".
- **FRU- and Management Controller Device Locator records:** FRU Information is associated to Entities and management controllers via the Entity ID and Entity Instance fields in FRU Device Locator records and Management Controller Device Locator records.
- **BMC Primary FRU Device information:** When an Management Controller Device locator record is not present (as is typical for the BMC) the FRU information in the Primary FRU Device is assumed to be associated with the FRU that the BMC itself is associated with. For example, if there's board information in the BMC Primary FRU, it is assumed that the information is for the board that the BMC is mounted on.

    - Chassis Info Area is FRU data associated with the overall chassis for the system. There should be only one set of Chassis Info Area data in the IPMI subsystem.
    - Board Info Area, implies existence of a board or module or other FRU. The name 'Board Info Area' is somewhat a misnomer, because the usage is not restricted to just circuit boards. This area is also typically used to provide FRU information for any replaceable entities, boards, andor sub-assemblies that are not sold as standalone products separately from other components. For example, individual boards from a board set, or a sub-chassis or backplane that's part of a larger chassis. If an explicit Entity ID and Entity Instance is not associated with the FRU Device, it is assumed that the information is for the main system board.
    - Product Info Area, if not associated with an explicit Entity via an Entity ID and Entity Instance, is assumed to be for the overall product. I.e. associated with the computer system.

If any Entity has associated sensors, but they're all disabled or inaccessible, do not instantiate the Entity, period. If that Entity is the top container Entity of an Entity Association, apply the remaining Entity instantiation rules as if the particular Entity Association did not exist in the SDRs at all.

Otherwise, if there is at least one explicit sensor that indicates that the top entity of an Entity Association is present, use the Entity association, and instantiate or do not instantiate the container and contained Entities based on the presence of the association.





2.8 Instantiating CIM Objects

2.8.1 Sensor Instantiation :

After discovering the available IPMI sensors, the next step is to instantiate CIM Objects based on the discovered sensors. This is done according to the Sensor Type information given in the SDR for the sensor. Which particular CIM Object(s) should be instantiated for a given sensor is the 'mapping' process is described in more detail in Section 3, IPMI Sensor Mapping, and following sections of this document.

2.8.1.1 Sensor state/reading rules

When populating attributes in CIM Objects that return state or reading information from a sensor, it is necessary to know when the IPMI sensor is returning valid information. This can generally be determined according to the following rules.

4. ignore sensor state/reading if the *reading/state unavailable* bit for the sensor is set.
5. if the sensor includes a 'presence' bit, ignore reading/state information for the sensor that requires the associated Entity to be present.
6. if a sensor that includes a 'presence' bit is associated with the container Entity of an Entity-Association AND the sensor state = "not present", ignore the reading/state information for the sensor that requires the associated Entity to be present, and ignore the reading/state of any sensors for contained Entities in the Entity association.
7. if an Entity Presence sensor is associated with the container Entity of an entity-association AND the sensor state = "not present", ignore the reading/state of any contained sensors in the Entity association.

Note:

Because IPMI specifies that disabled sensors (sensors that are set to ‘sensor scanning disabled’) should be ignored by software, it is recommended that instrumentation provider software does not disable sensors once they’ve been instantiated, since if the software was terminated, the sensors would remain disabled and a subsequent load of the software would ignore them.

Instrumentation software should not assume that the initial disabled state of a sensor (e.g. scanning enabled or disabled) will remain the same across restarts of the operating system, since it’s possible that a hardware change could have occurred between OS restarts that would cause the management controller to disable a sensor that had been previously enabled.

Refer to Section 3.4.1.2, Method: RequestStateChange for additional information.

2.8.2 Entity Instantiation

Once the IPMI Entities have been discovered, a CIM Objects should be instantiated according to the Entity ID for the Entity according to the steps described in Section 10, IPMI Entity Mapping. In addition to instantiating CIM Objects based on the Entities, CIM associations are created that relate those objects to one another, and to a ComputerSystemPackage association under the host Computer System.

2.8.3 FRU instantiation

IPMI FRU information is used as part of the Entity Instantiation process described in Section 10, IPMI Entity Mapping. In general, the instrumentation provider software should apply discovered FRU information as follows:

- Instantiate any FRU information associated with an instantiated Entity.

- Instantiate any FRU information associated with the BMC and any present satellite management controllers.
- Instantiate any FRU information for FRU devices listed in the SDRs and associated with an Entity that is otherwise unreferenced in the SDRs.
- It is possible that FRU information could be available for an Entity that is instantiated but not present (this is because FRU information can be kept in separate devices from the Entity). Software should check for this possibility and should assume that if the Entity is not present, FRU Information for the Entity is not valid and should be ignored.

3 IPMI Sensor Mapping

3.1 Introduction

IPMI Sensors can be mapped into the CIM model's CIM_Sensor class (and it's CIM_NumericSensor specialization). Discrete sensors become instances of CIM_Sensors. Analog sensors become instances of CIM_NumericSensor. IPMI entities, which are associated with IPMI Sensors in the IPMI Sensor Data Record (SDR), typically become instances of corresponding CIM_LogicalDevice's (or specializations thereof). The CIM_AssociatedSensor connects each instance of a sensor to the corresponding IPMI-Entity/CIM_LogicalDevice. Ultimately, all of these various class instances are associated with some enclosing instance of a CIM_ComputerSystem.

The following class diagram represents the various CIM classes that can be used when mapping IPMI Sensors to the Common Information Model (CIM). The AdminDomain, ComputerSystem and Logical Device classes are shown in this diagram to illustrate the way the Sensor classes are associated with overall class ecosystem. This is in preparation for interoperability with forthcoming (CIM) profiles from the DMTF and other standards bodies.

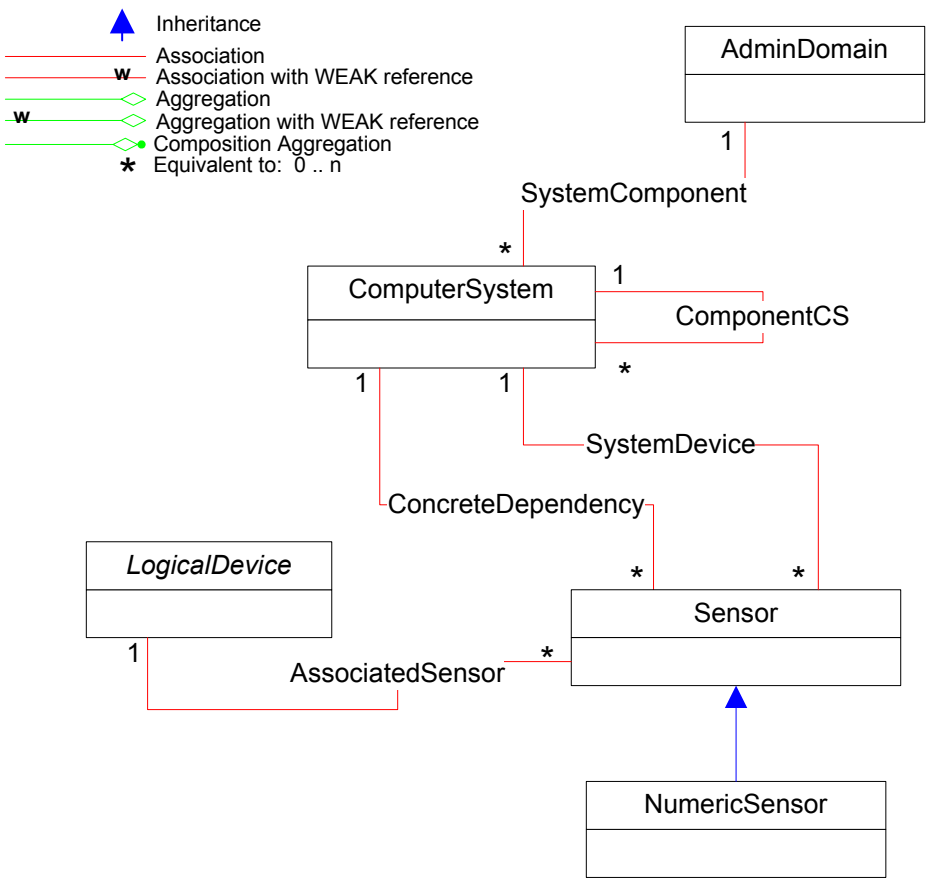


Figure 1 Class Diagram – IPMI Sensor Profile Mapping

3.2 Instance Examples

The following instance diagram represents that instantiation of a discrete voltage presence sensor (ie voltage is good or bad) and an analog temperature sensor (returns value in a numeric range). The AdminDomain, ComputerSystem and LogicalDevice instances are shown for illustration of how the sensor instances are associated to an overall object model representation of a managed system.

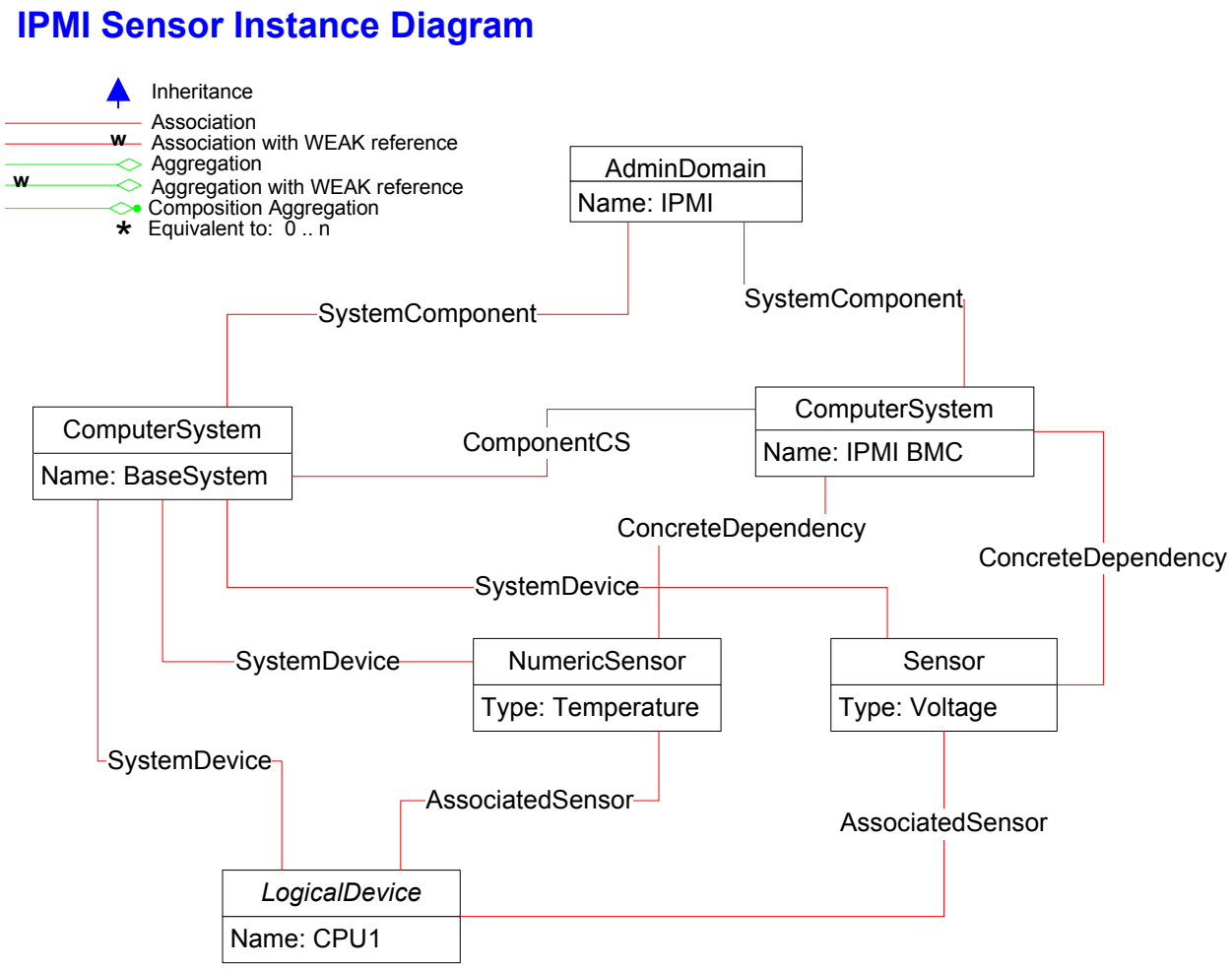


Figure 2 Instance Diagram – IPMI Sensor Profile Mapping

3.3 Mapping Requirements

3.3.1 Handling Sensor Specific Offsets

IPMI sensors that use sensor specific offsets present a CIM mapping challenge. There are no CIM classes that represent a sensor that itself represents multiple real-world sensors. Therefore,

mapping IPMI sensors with Sensor-specific Offsets must involve the generation an instance of CIM_Sensor that asserts each Sensor-specific offset. Technically, this is necessary only for those Sensor-specific Offsets which are enabled. The IPMI SDR Readable Mask, Assert Mask, or Deassert Mask describe just which Sensor-specific Offsets are enabled.

Instances of CIM_Sensor representing sensor specific offsets shall have a SensorType of "Other" and an OtherSensorTypeDescription set to a corresponding Event string value. The properly corresponding Event string values should be drawn, verbatim, from Table 36-3 of the IPMI 1.5 Specification.

The PossibleStates string array property shall be set to the following 2 strings:

1. <Event string value from Table 36-3> || "Assert"
2. <Event string value from Table 36-3> || "Deassert"

The CurrentState property shall have one of the 2 values of PossibleStates.

3.4 CIM_Sensor Mapping

3.4.1 CIM_Sensor Methods

3.4.1.1 Method: Reset()

```
uint32 Reset()
```

Per the DMTF definition: The return value should be 0 if the request was successfully executed, 1 if the request is not supported and some other value if an error occurred. For the IPMI mapping this method would cause the Re-arm Sensor Events Command to be invoked for the sensor.
Byte 2, bit 7 of the request data for the Rearm_Sensor_Events command should be set to 0 to rearm all events. Bytes 3-6 are not required to be sent.

Return value:
0 – Completed with no error
1 – Not Supported
2 – Unspecified Error
3 – Timeout
4 – Failed
5 – Invalid Parameter
6..0x0FFF – DMTF_Reserved
0x1000..0x7FFF – Method_Reserved
0x8000.. – Vendor_Reserved

3.4.1.2 Method: RequestStateChange

```
uint32 RequestStateChange (
   [IN] unit16 RequestedState,
   [IN] datetime TimeoutPeriod)
```

For the IPMI managed sensor, the current recommendation is that the method SHOULD NOT be supported. This recommendation is due to a sensor not being "discoverable" after being disabled. The current definition for discovering a sensor requires that an SDR be present for the sensor and the sensor enabled bit is on. If a sensor is dynamically disabled, subsequent "discovery" of sensors by IPMI CIM mapping implementations will not show the sensor.

Return value:
1 – Not Supported

3.4.2 CIM_Sensor Properties

REQUIRED properties are properties that MUST contain values supplied via the IPMI sub-system. RECOMMENDED properties are properties that MAY contain values supplied by the IPMI subsystem.

Table 1 CIM_Sensor Class Properties

Required Properties	Notes
SystemCreationClassName	**Key:**
SystemName	**Key:**
CreationClassName	**Key:**
DeviceID	**Key:**
SensorType	
OtherSensorTypeDescription	
Name	
CurrentState	
PossibleStates	
EnabledState	
HealthState	
OperationStatus	

Recommended Properties	Notes
PollingInterval	
StatusDescriptions	Not defined.
OtherIdentifyingInfo	Not defined.
IdentifyingDescriptions	Not defined.
AdditionalAvailability	Not defined.
InstallDate	Not defined.
RequestedState	Not defined.

Recommended Properties	Notes
TimeOfLastStateChange	Not defined.
OtherEnabledState	Not defined.
RequestedState	Not defined.
EnabledState	Not defined.
Caption	
Description	
ElementName	

3.4.2.1 CIM_Sensor.SystemCreationClassName

Per the DMTF definition of the class property: The scoping System's CreationClassName. In all cases this will be the class "CIM_ComputerSystem".

Formulate as a string:

<"CIM_ComputerSystem">

3.4.2.2 CIM_Sensor.SystemName

Formulate as a string:

"ComputerSystem.Name, of the ComputerSystem instance, across the SystemDevice association>

The CIM_ComputerSystem instance supplying the CIM_ComputerSystemName property value is the CIM_ComputerSystem instance the 'supervises' (or 'owns') the given CIM_Sensor instance. Typically, this CIM_ComputerSystem instance represents the management controller associated with the given sensor. This string value must be found and then copied in to the CIM_Sensor.SystemName property. Since there is but one instance of a management controller for most IPMI-compliant devices/systems, this is often quite easy to do.

Some modular systems may support an entire hierarchy of IPMI management controllers. For instance, the Advanced Telecommunications Architecture (ATCA – also known as PICMG 3.x), can support shelf management controllers (ShMCs). These are sometimes called Chassis Management Modules (CMMs – especially for bladed systems). Whatever the nomenclature, master management controllers often talk to some set of 'line' card or 'blade' card. Each such line/blade card can have its own embedded management controller. In turn, these (carrier) cards might support mezzanine cards (e.g. the ATCA mezzanine card standard). Once again, the mezzanine cards themselves might support IPMI management controllers (using names like the integrated platform management controller - IPMC). IPMI was designed to handle such hierarchical configurations of management controllers (and 'IPMI message bridging' amongst them).

In the presence of a management controller hierarchy, the CIM_Sensor.SystemName value comes from the particular instance of a CIM_ComputerSystem (representing a management controller) that has a direct CIM_SystemDevice association with the sensor in question.

3.4.2.3 CIM_Sensor.CreationClassName

As per the DMTF definition of this class property: The name of the class or the subclass used in the creation of an instance. This will typically be “CIM_Sensor” or “CIM_Numeric_Sensor”. However, some CIM implementations may include one or more proprietary subclasses of CIM_Sensor. In these cases the CIM_Sensor.CreationClassName value will be the name identifying such a subclass.

Formulate as a string:

”CIM_Sensor” or the name of the subclass created

3.4.2.4 CIM_Sensor.DeviceID

If SDR Record Type >= 01h and SDR Record Type <= 02h

The DeviceID is generated from the concatenation of the following fields separated by periods (".")

SDR Type (1|2)

Sensor Number (SDR type 1|2 table byte 8)

Owner LUN (SDR type 1|2 table byte 7)

Owner ID (SDR type 1|2 table byte 6)

Event/Reading Type Code (hex to string)

Sensor Specific Offset or "99" if none (hex to decimal to string)

Formulate as a string:

(string)(SensorNumber) || “.” || (string)(OwnerLUN) || “.” || (string)(OwnerID) ||

(string) (Event/Reading Type Code> || ”.” ||

(string) (Sensor Specific Offset or 99 if none)

Examples:

Discrete sensor, Full Record Typ 1, for Processor Sensor 6, LUN 0, Owner ID 1, Offset 07 (Presence Detected).

1.6.0.1.6F.7

Temperature sensor, Compact record Type 2, for Fan Sensor 8, LUN 0, Owner ID 2, no offset

2.8.0.2.01.99

else if SDR Record Type == C0h

The DeviceID is generated from the concatenation of the following fields separated by periods (".")

SDR Type (12)

SDR Record ID(SDR table byte 1 and 2)

Formulate as a string:

(string) 12 || (string) (SDR Record ID(SDR Table byte 1 and 2))

Example:

OEM SDR with Record ID 0102h

12.258

3.4.2.5 CIM_Sensor.ElementName and CIM_Sensor.Name

if SDR Record Type >= 01h and SDR Record Type <= 02h

Formulate as a string:

(string)<IPMI v1.5 SDR Record Type 01h or 02h, SDR entry bytes 49+, see IPMI v1.5 Table 37-1> || “(” || (string)(SensorNumber) || “.” || (string)(OwnerLUN) || “.” || string)(OwnerID) || “)”

Examples:

GeneralChassis1Intrusion(8.0.2)

DriveBay3Intrusion(7.0.1)

Processor4ThermalTrip(4.0.6)

LAN2Leash(8.0.8)

PUnit1OutputVoltage(6.0.1)

CPU1T(13.1.3)

else if SDR Record Type == C0h

Formulate as a string:

“OEM Record” || ”(“ || (string)(ManufacturerID) || “)”

3.4.2.6 CIM_Sensor.SensorType and CIM_Sensor.OtherSensorTypeDescription

When "present", all non-discrete IPMI sensors will be mapped to a corresponding instance of a NumericSensor. We say non-discrete, or analog, when we refer to sensors that provide a variable reading value (like a number of volts or a number of degrees). We say discrete when the sensor's reading values map to a fixed, a-priori enumerated set of values. These enumeration values are mapped by the IPMI specification to corresponding string values.

More specifically, any IPMI sensor whose Event/Reading Type code is 01h (i.e. Event/Reading Type code category == "threshold") is a non-discrete/analog sensor. See the IPMI v1.5 specification, section 36.1 (especially tables 36-1 and 36-2). In both type 1 and type 2 SDRs, the Event/Reading Type code is byte 14. See the IPMI specification's table 37-1 and table 37-2 for details.

The discrete versus analog sensor type distinction is *not* based on the IPMI Sensor Type Code (as per IPMI v1.5 specification Table 36-3). This might seem counter-intuitive, but this is the way IPMI works. Even though some IPMI Sensor Types do call for a specific, enumerated set of reading (offset) values, the Event/Reading Type code can override even these reserved values in favor of other Event/Reading Type code specific offset values (see Table 36-2). While overrides of this kind are extremely rare, they are possible. In the far more typical case, the Sensor (type) specific (offset) values are used. Details on the Sensor-Specific Offset appear in Table 36-3)

As mentioned, each analog sensor is mapped to a corresponding instance of a CIM_NumericSensor if, and *only if*, that particular sensor is *present*. Consult the Sensor Presence Analysis section above for the applicable presence detection algorithm (for any given IPMI SDR described sensor).

Formulate as a string:

if <Sensor is present> and <Sensor is not disabled>

  if <Event/ReadingTypeCode> (>= 01h and <=0Ch) or == 6Fh

    if SensorTypeCode == 01h
      CIM_Sensor.SensorType = CIM Temperature
    else if SensorTypeCode == 02h
      CIM_Sensor.SensorType = CIM Voltage
    else if SensorTypeCode == 03h
      CIM_Sensor.SensorType = CIM Current (e.g. amps)

    else if SensorTypeCode == 04h

      CIM_Sensor.SensorType = CIM Tachometer

    else if SensorTypeCode == 05h

      CIM_Sensor.SensorType = CIM Other

      CIM_Sensor.OtherSensorTypeDescription = "ChassisIntrusion"

```
1254        else if SensorTypeCode == 06h
1255            CIM_Sensor.SensorType = CIM Other
1256            CIM_Sensor.OtherSensorTypeDescription = “PlatformSecurityViolation“
1257        else if SensorTypeCode == 07h
1258            CIM_Sensor.SensorType = CIM Other
1259            CIM_Sensor.OtherSensorTypeDescription = “Processor“
1260        else if SensorTypeCode == 08h
1261            CIM_Sensor.SensorType = CIM Other
1262            CIM_Sensor.OtherSensorTypeDescription = “PowerSupply“
1263        else if SensorTypeCode == 09h
1264            CIM_Sensor.SensorType = CIM Other
1265            CIM_Sensor.OtherSensorTypeDescription = “PowerUnit“
1266        else if SensorTypeCode == 0Ah
1267            CIM_Sensor.SensorType = CIM Other
1268            CIM_Sensor.OtherSensorTypeDescription = “CoolingDevice“
1269        else if SensorTypeCode == 0Bh
1270            CIM_Sensor.SensorType =  CIM Other
1271            CIM_Sensor.OtherSensorTypeDescription = “UnitsBasedSensor”
1272        else if SensorTypeCode == 0Ch
1273            CIM_Sensor.SensorType = CIM Other
1274            CIM_Sensor.OtherSensorTypeDescription = “Memory“
1275        else if SensorTypeCode == 0Dh
1276            CIM_Sensor.SensorType = 1  # CIM Other
1277            CIM_Sensor.OtherSensorTypeDescription = “DriveSlotOrBay”
1278        else if SensorTypeCode == 0Eh
1279            CIM_Sensor.SensorType = 1  # CIM Other
```

1280 CIM_Sensor.OtherSensorTypeDescription = “POSTMemoryResize”

1281 else if SensorTypeCode == 0Fh

1282 CIM_Sensor.SensorType = CIM Other

1283 CIM_Sensor.OtherSensorTypeDescription = “SystemFirmwarePost“

1284 else if SensorTypeCode == 10h

1285 CIM_Sensor.SensorType = CIM Other

1286 CIM_Sensor.OtherSensorTypeDescription = “EventLoggingDisabled“

1287 else if SensorTypeCode == 11h

1288 CIM_Sensor.SensorType = CIM Other

1289 CIM_Sensor.OtherSensorTypeDescription = “Watchdog1“

1290 else if SensorTypeCode == 12h

1291 CIM_Sensor.SensorType = CIM Other

1292 CIM_Sensor.OtherSensorTypeDescription = “SystemEvent“

1293 else if SensorTypeCode == 13h

1294 CIM_Sensor.SensorType = CIM Other

1295 CIM_Sensor.OtherSensorTypeDescription = “CriticalInterrupt“

1296 else if SensorTypeCode == 14h

1297 CIM_Sensor.SensorType = CIM Other

1298 CIM_Sensor.OtherSensorTypeDescription = “Button“

1299 else if SensorTypeCode == 15h

1300 CIM_Sensor.SensorType = CIM Other

1301 CIM_Sensor.OtherSensorTypeDescription = “ModuleOrBoard”

1302 else if SensorTypeCode == 16h

1303 CIM_Sensor.SensorTypeCode = CIM Other

1304 CIM_Sensor.OtherSensorTypeDescription = “MicroControllerOrCoProcessor”

1305 else if SensorTypeCode == 17h

1306 CIM_Sensor.SensorTypeCode = CIM Other

1307 CIM_Sensor.OtherSensorTypeDescription = “AddInCard”

1308 else if SensorTypeCode == 18h

1309 CIM_Sensor.SensorTypeCode = CIM Other

1310 CIM_Sensor.OtherSensorTypeDescription = “Chassis“

1311 else if SensorTypeCode == 19h

1312 CIM_Sensor.SensorTypeCode = CIM Other

1313 CIM_Sensor.OtherSensorTypeDescription = “Chip Set“

1314 else if SensorTypeCode == 1Ah

1315 CIM_Sensor.SensortTypeCode = CIM Other

1316 CIM_Sensor.OtherSensorTypeDescription = “OtherFRU”

1317 else if SensorTypeCode == 1Bh

1318 CIM_Sensor.SensorTypeCode = CIM Other

1319 CIM_Sensor.OtherSensorTypeDescription = “CableOrInterconnect”

1320 else if SensorTypeCode == 1Ch

1321 CIM_Sensor.SensorTypeCode = CIM Other

1322 CIM_Sensor.OtherSensorTypeDescription = “Terminator”

1323 else if SensorTypeCode == 1Dh

1324 CIM_Sensor.SensorType = CIM Other

1325 CIM_Sensor.OtherSensorTypeDescription = “SystemBootInitiated“

1326 else if SensorTypeCode == 1Eh

1327 CIM_Sensor.SensorType = CIM Other

1328 CIM_Sensor.OtherSensorTypeDescription = “BootError“

1329 else if SensorTypeCode == 1Fh

1330 CIM_Sensor.SensorType = CIM Other

1331 CIM_Sensor.OtherSensorTypeDescription = “OSBoot“

```
1332        else if SensorTypeCode == 20h
1333          CIM_Sensor.SensorType = CIM Other
1334          CIM_Sensor.OtherSensorTypeDescription = “OSCriticalStop“
1335        else if SensorTypeCode == 21h
1336          CIM_Sensor.SensorType = CIM Other
1337          CIM_Sensor.OtherSensorTypeDescription = “Slot/Connector“
1338        else if SensorTypeCode == 22h
1339          CIM_Sensor.SensorType = CIM Other
1340          CIM_Sensor.OtherSensorTypeDescription = “SystemACPIPowerState“
1341        else if SensorTypeCode == 23h
1342          CIM_Sensor.SensorType = CIM Other
1343          CIM_Sensor.OtherSensorTypeDescription = “WatchDog2“
1344        else if SensorTypeCode == 24h
1345          CIM_Sensor.SensorType = CIM Other
1346          CIM_Sensor.OtherSensorTypeDescription = “PlatformAlert“
1347        else if SensorTypeCode == 25h
1348          CIM_Sensor.SensorType = CIM Other
1349          CIM_Sensor.OtherSensorTypeDescription = “EntityPresence“
1350       else if SensorTypeCode == 26h
1351          CIM_Sensor.SensorTypeCode =  CIM Other
1352          CIM_Sensor.OtherSensorTypeDescription = “MonitorASIC-IC”
1353       else if SensorTypeCode >= C0h and SensorTypeCode <= FFh
1354          CIM_Sensor.SensorType =  CIM Other
1355          CIM_Sensor.OtherSensorTypeDescription = “OEM”
1356        else
1357          #  This sensor does not map to an instance of
```

```
1358          #  NumericSensor.  It also does not map to an
1359          #  instance of a Sensor nor any specialization of
1360          #  CIM_Sensor.  This sensor is ignored in the IPMI-to-CIM
1361          #  mapping.
1362       else   #  This sensor's Event/Reading Type Code != 01h
1363       #  Thus, this sensor will become an instance of Sensor (see below)
1364    else  #  Either this sensor is not present or this sensor is disabled
1365    #  Skip this sensor.  It will not map to any CIM object instance.
```

1366 3.4.2.7 CIM_Sensor.PossibleStates

1367 Formulate as a string:

```
1368    if <Event/ReadingTypeCode> >= 01h and <Event/ReadingTypeCode> <= 0Ch
1369       #  Get PossibleStates[] strings from Table 36-2.  These Event/ReadingTypeCodes
1370       #  use the "Generic States".  See section 36.1 for details.
1371    else if <Event/ReadingTypeCode> == 6Fh  #  The Sensor Specific Event/Reading Type
1372       #  As per the intent of section 3.3.1, Handling Sensor Specific Offsets
1373       for  evs in EStr(<given the Sensor Type Code, get all corresponding Sensor-specific Offsets>)
1374          cur_sensor = #create new instance of CIM_Sensor
1375          curr_sensor.PossibleStates[0] = evs || " Assert"  # The assert/deassert order is critical
1376                                                            #  to section 3.4.2.8
1377          curr_sensor.PossibleStates[1] = evs || " Deassert"
1378               See Section 3.3.1 Handling Sensor Specific Offsets
1379
1380    else if <Event/ReadingTypeCode> >= 70h and <Event/ReadingTypeCode> <= 7Fh
1381       #  PossibleStates will be undefined, as discrete state information is specific to the OEM
1382       #  identified by the Manufacturer ID for the IPM device that is providing access to the
1383       #  sensor.  Software needing Manufacturer ID can call the GetManufactuererID method to
1384       # obtain this information
1385    else
1386        #  This is not an Event/ReadingTypeCode supported by the IPMI-to-CIM mapping.
```

Skip this sensor. It will not map to a CIM object instance.

3.4.2.8 CIM_Sensor.CurrentState

If the IPMI GetSensorReading command returns a byte 4, then map the bit

fields of byte 4 to one of the CIM_NumericSensor.PossibleStates above.

If not, then the CurrentState is "OK".

If multiple bits are set in byte 4, See Section 3.3.1 Handling Sensor Specific Offsets.

```
# Given the IPMI sensor identifier,
# and the Get Sensor Reading Command byte 4/5 response,
# reverse map to find the corresponding instances (plural) of CIM_Sensor.
#    Remember that a single IPMI sensor with Sensor Specific Offsets
#    can turn into a collection of CIM_Sensor instances – one for asserting
#    each sensor specific offset.  I'll call the associated CIM_Sensor instances
#    related CIM Sensors.
#
States_Asserted[] = <byte 4 [7:0] from Get Sensor Reading Command &
                                Byte 5 [6:0] from Get Sensor Reading Command>
Related_Sensors[] = sort-by-IPMI-offset<related_CIM_Sensors>
i=0;
for current_sensor in <Related_Sensors>
  if Sates_Asserted[i]
    current_sensor.CurrentState = current_sensor.PossibleState[0] # Assert
  else
    current_sensor.CurrentState = current_sensor.PossibleState[1] # Deassert
```

3.4.2.9 CIM_Sensor.Caption

if SDR Record Type >= 01h and SDR Record Type <= 02h

Formulate as a string:

(string)TruncAt(64, EStr(CIM_Sensor.SensorType)) || "(" || (string)(SensorNumber) || "."
|| (string)(OwnerLUN) || "." || (string)(OwnerID) || ")")

Example:

Voltage(6.0.1)

Temperature(13.1.3)

else if SDR Record Type == C0h

Formulate as a string:

1421 “OEM Record” || ”(“ || (string)(ManufacturerID) || “)”

1422 #### 3.4.2.10 CIM_Sensor.Description

1423 Formulate as a string:

1424 if SDR Record Type >= 01h and SDR Record Type <= 02h

1425 (string)<IPMI v1.5 SDR Record Type 01h or 02h, SDR entry bytes 49+, see IPMI v1.5
1426 Table 37-1> || “(” || (string)(SensorNumber) || “.” || (string)(OwnerLUN) || “.” ||
1427 string)(OwnerID) || “): ” ||(string)(EStr(<CIM_Sensor.SensorType>)) || “ for ” ||
1428 (string)(<IPMI_SDR_EntityID>) || “ “ || (string)(<IPMI_SDR_EntityInstanceNumber>)

1429 else if SDR Record Type == C0h

1430 (string) “OEM Record (“|| (string)(ManufacturerID) “) : SDR Type C0h - OEM Data” ||
1431 (uint8[56])(SDR entry bytes 9-64, see IPMI v1.5 Table 37.11)

1432 Example:

1433 PUnitVolt1(6.0.1): Voltage for Power Supply 1

1434 CPU1T(13.1.3): Temperature for processor 1

1435 OEM Record (343): SDR Type C0h - OEM Data (0Dh FBh 7Bh D0h 00h 00h 00h 00h 00h
1436 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h
1437 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h
1438 00h 00h 00h)

1439 #### 3.4.2.11 CIM_Sensor.PollingInterval

1440 Format as a string:

1441 The polling interval that the Sensor hardware or the instrumentation uses to determine the current
1442 state of the Sensor. Units are nanoseconds.

1443 #### 3.4.2.12 CIM_Sensor.HealthState

1444 TBD. Currently being profiled in DMTF SMWG.

1445 #### 3.4.2.13 CIM_Sensor.OperationalStatus

1446 TBD. Currently being profiled in DMTF SMWG.

1447 ## 3.5 CIM_NumericSensor Mapping

1448 CIM_NumericSensor is subclassed from CIM_Sensor and inherits many properties. The
1449 properties in Table 2 CIM_NumericSensor Class Properties that are the same as for an instance
1450 of CIM_Sensor MUST be implemented as defined in the “See <property name>” reference in the
1451 Notes column for each property.

3.5.1 CIM_NumericSensor.Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that listed in the Required section in Table 2 below MUST be implemented as defined in the subsection following this table unless the Notes column for each property specifies the defining reference.

Table 2 CIM_NumericSensor Class Properties

Required Properties	Notes
SystemCreationClassName	**Key:** See CIM_Sensor.SystemCreationClassName mapping
SystemName	**Key:** See CIM_Sensor.SystemName Mapping
CreationClassName	**Key:** See CIM_Sensor.CreationClassName mapping
DeviceID	**Key:** See CIM_Sensor.DeviceID mapping
SensorType	See CIM_Sensor.SensorType mapping
OtherSensorTypeDescription	See CIM_Sensor.OtherSensorTypeDescription mapping
Name	See CIM_Sensor.SystemCreationClassName mapping
HealthState	See CIM_Sensor.HealthStatus mapping
OperationStatus	See CIM_Sensor.OperationStatus mapping
CurrentReading	
BaseUnits	
UnitModifier	
RateUnits	

Recommended Properties	Notes
NominalReading	
NormalMax	
NormalMin	
MaxReadable	
MinReadable	
Resolution	
Accuracy	
IsLinear	
Hysterisis	
SupportedThresholds	
EnableThresholds	
SettableThresholds	
PollingInterval	See CIM_Sensor.PollingInterval
PossibleStates	See CIM_Sensor.PossibleStates mapping

Recommended Properties	Notes
CurrentState	See CIM_Sensor.CurrentState mapping
Caption	See CIM_Sensor.Caption mapping
Description	See CIM_Sensor.Description mapping
ElementName	See CIM_Sensor.ElementName mapping
StatusDescriptions	Not defined.
LowerThresholdNonCritical	
UpperThresholdNonCritical	
LowerThresholdCritical	
UpperThresholdCritical	
LowerThresholdFatal	
UpperThresholdFatal	
RequestedState	Not defined.
EnabledState	Not defined.
TimeOfLastStateChange	Not defined.
OtherEnabledState	Not defined.

3.5.1.1 NumericSensor.CurrentReading

The current value indicated by the Numeric Sensor.

3.5.1.2 NumericSensor.UnitModifier, BaseUnits, And RateUnits

We must do more than just map IPMI sensor's with an obviously matching IPMI Sensor Unit Type Code. An obvious match would be IPMI 'seconds' matching to CIM_Sensor.BaseUnits == 'seconds'. Scaling issues are less obvious. For example, IPMI 'milliseconds' must map to CIM_Sensor.BaseUnits = 'seconds' with a corresponding CIM_Sensor.UnitModfier = -3. The scaling required might sometimes be a bit less trivial. Still, all IPMI "BaseUnits" must be mapped to corresponding CIM "BaseUnits".

CIM represents floating point values as signed integers (sint32) that are scaled. IPMI tends to use floating point numeric representations (e.g. IEEE 754/854). To map these into signed integers, we'll just assume a default "precision" value of "-2" (for the CIM member "UnitModifier"). We'll round the IPMI floating point value appropriately and then convert it to integer required. Recall that a CIM UnitModifer of -2 means that the CIM CurrentReading will be will multiplied by 0.01 (10^{-2}). So a CurrentReading of 333, where the CIM BaseUnits is "Voltage", means that the true current reading is 3.33v.

The RateUnits from the IPMI SDR match one-to-one with the enumerated CIM_Sensor.RateUnits. The last three CIM_Sensor.RateUnits are not used by IPMI (e.g. "per week", "per month", "per year").

Examples: BaseUnits

EStr(5) == "Volts"

EStr(3) == “Degrees F”

Examples: RateUnits

EStr(0) == “None”

EStr(1) == “Per Microsecond”

EStr(2) == “Per Millisecond

EStr(3) == “Per Second”

EStr(4) == “Per Minute”

EStr(5) == “Per Hour”

EStr(6) == “Per Day”

3.5.1.3 NumericSensor.NominalReading

NominalReading indicates the 'normal' or expected value for the NumericSensor.

3.5.1.4 NumericSensor.NormalMax

NormalMax provides guidance for the user as to the normal maximum range for the NumericSensor

3.5.1.5 NumericSensor.NormalMin

NormalMin provides guidance for the user as to the normal minimum range for the NumericSensor.

3.5.1.6 NumericSensor.MaxReadable

MaxReadable indicates the largest value of the measured property that can be read by the NumericSensor.

3.5.1.7 NumericSensor.MinReadable

MinReadable indicates the smallest value of the measured property that can be read by the NumericSensor.

3.5.1.8 NumericSensor.Resolution

Resolution indicates the ability of the Sensor to resolve differences in the measured property. The units for this measurement are determined by "BaseUnit*UnitModifier/RateUnit.

3.5.1.9 NumericSensor.Accuracy

Indicates the accuracy of the Sensor for the measured property. Its value is recorded as plus/minus hundredths of a percent. Accuracy, along with Resolution, is used to calculate the actual value of the measured physical property. Accuracy may vary depending on whether the Device is linear over its dynamic range.

Example value: 0

3.5.1.10 NumericSensor.IsLinear

Indicates that the Sensor is linear over its dynamic range.

Example value: TRUE

3.5.1.11 NumericSensor.Hysteresis

Indicates the margin built around the thresholds. This margin prevents unnecessary state changes when the Sensor reading may fluctuate very close to its thresholds. This could be due to the Sensor's tolerance/accuracy/resolution or due to environmental factors. Once a threshold is crossed, the state of the Sensor should change. However, the state should not fluctuate between the old and new states unless the Sensor's change in the reading exceeds the hysteresis value. The units for this measurement are determined by BaseUnit*UnitModifier/RateUnit.

Fixed value of: 0

3.5.1.12 NumericSensor.SupportedThresholds

If supported by BMC (from SDR).

3.5.1.13 NumericSensor.EnableThresholds

If supported by BMC (from SDR).

3.5.1.14 NumericSensor.SettableThresholds

If supported by BMC (from SDR).

3.5.1.15 NumericSensor.LowerThresholdNonCritical

The Sensor's threshold values specify the ranges (min and max values) for determining whether the Sensor is operating under Normal, NonCritical, Critical or Fatal conditions. If Current Reading is between LowerThresholdNonCritical and Upper ThresholdNonCritical, then the Sensor is reporting a normal value. If CurrentReading is between LowerThresholdNonCritical and LowerThresholdCritical, then the CurrentState is NonCritical.

Example value: 50

3.5.1.16 NumericSensor.UpperThresholdNonCritical

The Sensor's threshold values specify the ranges (min and max values) for determining whether the Sensor is operating under Normal, NonCritical, Critical or Fatal conditions. If the CurrentReading is between LowerThresholdNonCritical and UpperThresholdNonCritical, then the Sensor is reporting a normal value. If the CurrentReading is between UpperThreshold NonCritical and UpperThresholdCritical, then the CurrentState is NonCritical.

Example value: 500

3.5.1.17 NumericSensor.LowerThresholdCritical

The Sensor's threshold values specify the ranges (min and max values) for determining whether the Sensor is operating under Normal, NonCritical, Critical or Fatal conditions. If the CurrentReading is between LowerThresholdCritical and Lower ThresholdFatal, then the CurrentState is Critical.

Example value: -1200

3.5.1.18 NumericSensor.UpperThresholdCritical

The Sensor's threshold values specify the ranges (min and max values) for determining whether the Sensor is operating under Normal, NonCritical, Critical or Fatal conditions. If the CurrentReading is between UpperThresholdCritical and Upper ThresholdFatal, then the CurrentState is Critical.

Example value: 550

3.5.1.19 NumericSensor.LowerThresholdFatal

The Sensor's threshold values specify the ranges (min and max values) for determining whether the Sensor is operating under Normal, NonCritical, Critical or Fatal conditions. If the CurrentReading is below LowerThresholdFatal, then the Current State is Fatal.

Example value: -1279

3.5.1.20 NumericSensor.UpperThresholdFatal

The Sensor's threshold values specify the ranges (min and max values) for determining whether the Sensor is operating under Normal, NonCritical, Critical or Fatal conditions. If the CurrentReading is above UpperThresholdFatal, then the Current State is Fatal.

Example value: 1270

4 IPMI Log Mapping

4.1 Introduction

The IPMI Log Mapping extends the management capability of the IPMI to CIM mapping by adding the capability to access and manage the IPMI System Event Log (SEL) entries via the CIM_RecordLog and CIM_LogRecord classes.

The following class diagram describes the CIM classes involved in representing IPMI System Event Log by the CIM_RecordLog class and an individual SEL entries with the CIM LogRecord class. The AdminDomain and ComputerSystem classes are shown in this diagram to illustrate the way the RecordLog class is associated with overall class ecosystems. This is in preparation for interoperability with forthcoming profiles from the DMTF and other standards bodies.

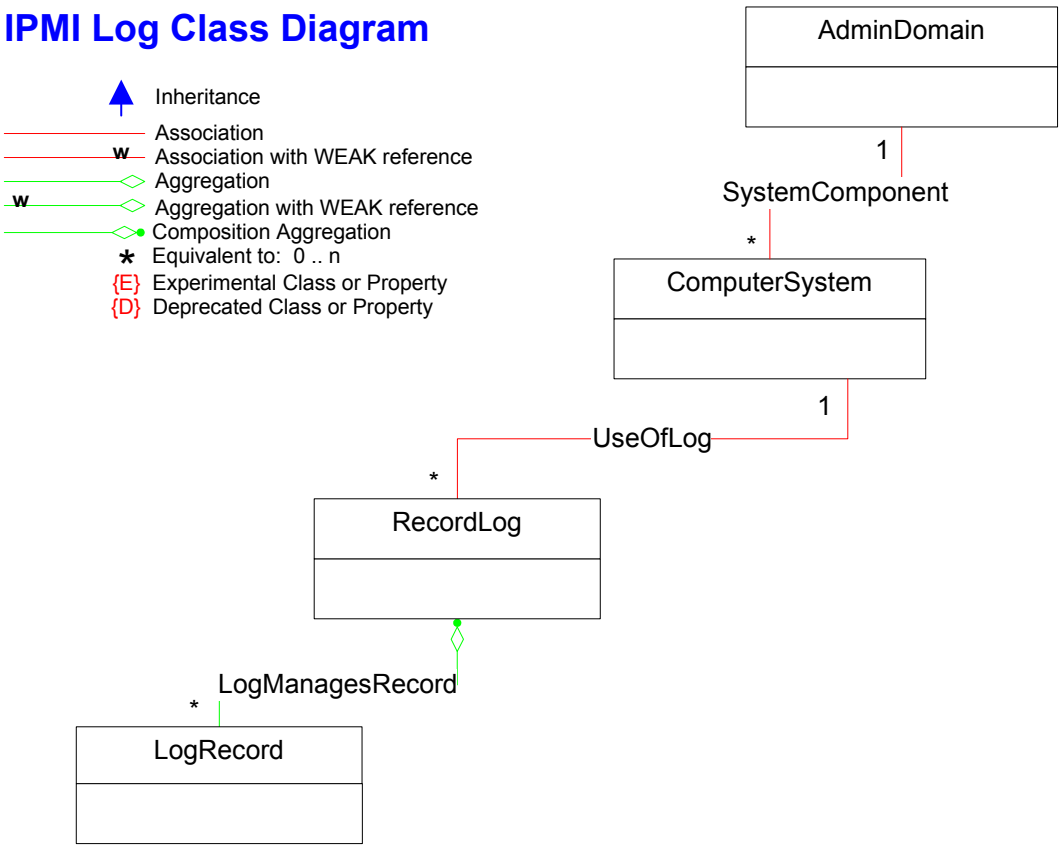


Figure 3 Class Diagram - IPMI Log

Note: The CIM_UseOfLog and CIM_LogManagesRecord associations are changed from previous versions of the IPMI CIM Mapping Guideline due to changes in the DMTF Record Log Profile.

4.2 Instance Diagrams

The IPMI Log Mapping is the CIM object model to IPMI interface mapping needed to manage the IPMI SEL (Sytem Event Log) exposed by an embedded Baseboard Management Controller or Service Processor with IPMI compliant firmware. The objects modeled in this profile are related to the IPMI BMC SEL and SEL entries specific to the IPMI Specification.

The following instance diagram represents that instantiation of 2 SEL entries as LogRecord instances and the SEL log by a RecordLog instance. The RecordLog instance has the methods used to manipulate the SEL (ex. ClearLog). The AdminDomain and ComputerSystem instances are shown for illustration of how the SEL log and entries instances are associated to an overall object model representation of a managed system.

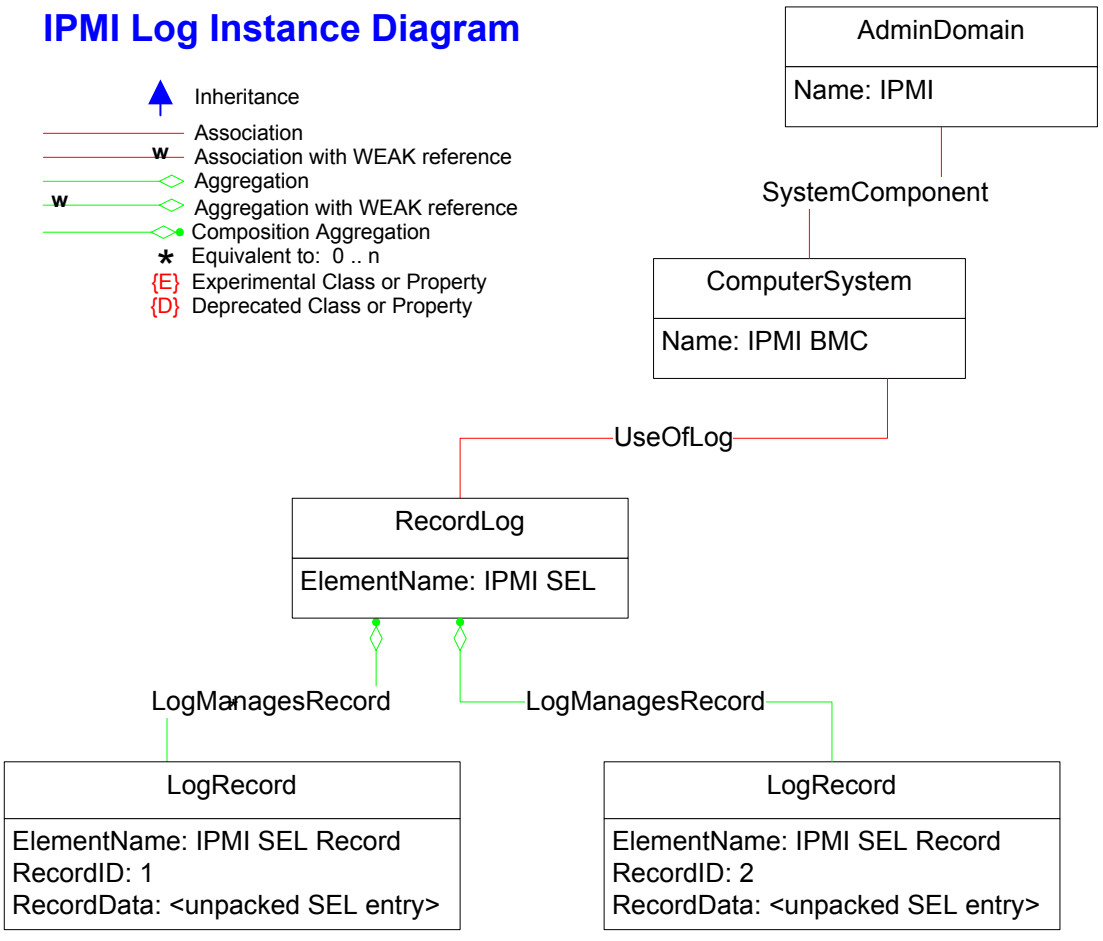


Figure 4 Instance Diagram – IPMI Log

4.3 Mapping Requirements

4.3.1 Associating SEL with SDR

A SEL record's associated CIM_Sensor instance (whether that is a NumericSensor or a Sensor instance) can be found using the SEL Record's GeneratorID (bytes 8-9) and Sensor Number

(byte 12). See Table 26-1 for details. This SEL data maps to the CIM_Sensor.DeviceID key value (as defined in the IPMI-to-CIM sensor mapping document). Once this Sensor key string is extracted from the SEL entry in question, the corresponding CIM_Sensor instance can be found.

Behind the scenes, the corresponding CIM_Sensor maps to a corresponding IPMI SDR record (or either type 1 or type 2). This indirectly corresponding SDR record must also be found in order to extract the IPMI_SDR_EntityID and IPMI_SDR_EntityInstanceNumber mentioned above.

We refer to the CIM_Sensor instance above because CIM_Sensor is a common base class for both NumericSensor and Sensor (as “trivially derived” for this IPMI-to-CIM mapping).

4.4 CIM_RecordLog Mapping

4.4.1 CIM_RecordLog.Methods

4.4.1.1 Method: ClearLog

```
uint32 ClearLog()
```

MUST delete all SEL records:

An application clearing the SEL should utilize the recommended algorithm described below:

a) first check to see if SEL is empty by checking the number of records using "Get SEL Info"

b) if already empty, you're done

c) if SEL not empty, use "Clear SEL" to see if erase is already in progress (in case Get SEL Info doesn't return correct status)

d) if erase already in progress then wait (and potentially time out) and exit

e) otherwise (if SEL is not empty) issue the Clear SEL command, then wait (and potentially time out) and exit

Note: Originally, the ClearLog method on the MessageLog class would issue an IPMI request to clear the log and return success. But the log may not have actually have cleared by the time success is returned. The change made was to issue the clear request and spin for sometime checking if the operation was still in progress. After some period, the operation will timeout and a timeout error is returned to the client. This means the client needs to call ClearLog again (and also spin). This scenario is addressed by having the application utilize the algorithm described above.

Return value:
0 – Completed with no error
1 – Not Supported
2 – Unspecified Error
3 – Timeout
4 – Failed
5 – Invalid Parameter

6..0x0FFF – DMTF_Reserved
0x1000..0x7FFF – Method_Reserved
0x8000.. – Vendor_Reserved

4.4.1.2 Method: RequestStateChange

```
uint32 RequestStateChange (
   [IN] unit16 RequestedState,
   [IN] datetime TimeoutPeriod)
```

For the IPMI SEL log, the action this method takes on the log is to freeze or stop the log if the log from accepting further input if the RequestedState parameter is "Disabled". If the RequestedState parameter is "Enabled", the IPMI SEL log is unfrozen or will continue to accept log entries in the normal fashion. Not all IPMI implementations support the freezing of the IPMI SEL log and these implementation will need to return "Not Supported" if the RequestStateChange method is invoked.

Return value:

0 – Completed with no error
1 – Not Supported
2 – Unspecified Error
3 – Timeout
4 – Failed
5 – Invalid Parameter
6..0x0FFF – DMTF_Reserved
0x1000..0x7FFF – Method_Reserved
0x8000.. – Vendor_Reserved

4.4.2 CIM_RecordLog.Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that listed in the Required section in Table 2 below MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

Table 3 CIM_RecordLog Class Properties

Required Properties	Notes
InstanceID	**Key:**
Name	
MaxNumberOfRecords	
CurrentNumberOfRecords	
EnabledState	
OperationalStatus	
HealthState	

Recommended Properties	Notes
OtherEnabledState	Not defined.
RequestedState	Not defined.
EnabledDefault	Not defined.
TimeOfLastStateChange	Not defined.
InstallDate	Not defined.
StatusDescriptions	Not defined.
Caption	
Description	
ElementName	

The following subsections describe procedures for generating values for various properties of CIM classes required in this profile.

4.4.2.1 CIM_RecordLog.InstanceID

Per the DMTF definition: In order to ensure uniqueness within the NameSpace, the value of InstanceID SHOULD be constructed using the following ' preferred ' algorithm: < OrgID > : < LocalID > Where < OrgID > and < LocalID > are separated by a colon ' : '.

Formulate as a string:

> <"IPMI" || ":" || < ComputerSystem.Name of the ComputerSystem instance of the one associated BMC > || " SEL Log">

If there are several BMC's in the same namespace, unique names for the BMC's need to be generated in order to distinguish between the different instantiations of RecordLog. The name of the BMC can be formulated by instrumentation using unique identifying values obtained from the BMC or derived from intrumentation access deviations.

4.4.2.2 CIM_RecordLog.Name

Fixed value of "IPMI SEL"

4.4.2.3 CIM_RecordLog.MaxNumberOfRecords

Calculated by instrumentation using the following approach:

The maximum number of SEL records can be computed by issuing the Get_Sel_Allocation command and using the response data. Dividing the number of free allocation units (bytes 4,5) by the maximum record size in allocation units (byte 10) will provide the maximum number of records. In the event that Get_Sel_Allocation command is not supported, the value should be set to 0.

4.4.2.4 CIM_RecordLog. CurrentNumberOfRecords

Calculated by instrumentation using the following approach:.

The current number of log records can be determined by issuing a Get_Sel_Info command and examining the number of log entries (bytes 3,4) in the response data

4.4.2.5 CIM_RecordLog.EnabledState

Per definition in CIM mof: EnabledState is an integer enumeration that indicates the enabled/disabled states of an element. For the IPMI mapping, the value for EnabledState can only be "Enabled" and "Disabled". Enabled indicates the SEL is active and accepting new entries. Disabled indicates the log is "frozen" or full. Frozen is when the SEL is programmatically suspended to not accept further entries and is available in some IPMI implementations (See the RequestStateChange method definition).

Formulate as an integer:

2 – Enabled

3 - Disabled

4.4.2.6 CIM_RecordLog.HealthState

TBD. Currently being profiled in DMTF SMWG.

4.4.2.7 CIM_RecordLog.OperationalStatus

TBD. Currently being profiled in DMTF SMWG.

4.4.2.8 CIM_RecordLog. Caption

Fixed value of "IPMI SEL"

4.4.2.9 CIM_RecordLog. Description

Fixed value of "IPMI SEL"

4.4.2.10 CIM_RecordLog. ElementName

Fixed value of "IPMI SEL"

4.5 CIM_LogRecord Mapping

4.5.1 CIM_LogRecord Methods

None Defined.

4.5.2 CIM_LogRecord Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that listed in the Required section in Table 2 below MUST be implemented as defined in the subsection following this table unless the Notes column for each property specifies the defining reference.

Table 4 CIM_LogRecord Class Properties

Required Properties	Notes
LogCreationClassName	**Key:**
LogName	**Key:**
CreationClassName	**Key:**
RecordID	**Key:**
MessageTimestamp	**Key:**
RecordFormat	
RecordData	

Recommended Properties	Notes
Caption	
Description	
ElementName	

4.5.2.1 CIM_LogRecord.LogCreationClassName

Per the DMTF definition of the class property: The name of the class or the subclass used in the creation of an instance of the log for with this record is a part. This will typically be "CIM_RecordLog".

Formulate as a string:

<"CIM_RecordLog" or the name of the subclass created for the containing log>

4.5.2.2 CIM_LogRecord.LogName

Propagated ("CIM_RecordLog.Name")

Fixed value of: "IPMI SEL"

4.5.2.3 CIM_LogRecord.CreationClassName

Per the DMTF definition of the class property: The name of the class or the subclass used in the creation of an instance. This will typically be "CIM_LogRecord".

Formulate as a string:

<”CIM_LogRecord”>

4.5.2.4 CIM_LogRecord.RecordID

RecordID is a Key and must not be Null. The value will be populated with the IPMI SEL Record ID. Used in conjunction with the MessageTimestamp, must uniquely identify an instance of LogRecord that is associated via RecordInLog to the containing instance of RecordLog representing the SEL.

Formulate as a string:

<SEL Record ID as passed by the Get_SEL_Entry IPMI command>

4.5.2.5 CIM_LogRecord. MessageTimestamp

This value is the timestamp of the SEL entry plus an offset to GMT. There is an assumption that the BMC time will be set by the system management software running “in-band” in the managed system that the BMC resides. In band instrumentation will generate the GMT offset required by this data type by discerning the timezone that the managed system is in and producing the necessary offset. Out-of-band instrumentation doing CIM mapping will need to determine the difference with the BMC time and add that difference to the offset for the timezone the out-of-band client is in.

Formulate as a string using the datetime data type as defined by the DMTF:

Example value: “20040924152230.000000+480”

For non-timestamped OEM SEL, this property is set to ”Unknown” as defined by the DMTF:

Value = “99990101000000.000000+000”

This definition requires that the BMC time MUST be set by in-band instrumentation to the time of the owning managed system. This is typically done today and is a Best Practice.

4.5.2.6 CIM_LogRecord.RecordFormat

A free-form string describing the LogRecord's data structure. To describe the data structure of RecordData, the RecordFormat string should be constructed as follows: The first character is a delimiter character and is used to parse the remainder of the string into sub-strings. Each sub-string is separated by the delimiter character and should be in the form of a CIM property declaration (i.e., datatype and property name). This set of declarations may be used to interpret the similarly delimited RecordData property

For system events, value of:
"*string IPMI_SensorNumber.IPMI_OwnerLUN.IPMI_OwnerID*uint8[2]
IPMI_RecordID*uint8 IPMI_RecordType*uint8[4] IPMI_Timestamp*uint8[2]
IPMI_GeneratorID*uint8 IPMI_EvMRev*uint8 IPMI_SensorType*uint8

IPMI_SensorNumber*boolean IPMI_AssertionEvent*uint8 IPMI_EventType*uint8 IPMI_EventData1*uint8 IPMI_EventData2*uint8 IPMI_EventData3*uint32 IANA*"

For timestamped OEM events, value of:
"*uint8[2] IPMI_RecordID*uint8 IPMI_RecordType*uint8[4] IPMI_Timestamp*uint8[3] IPMI_ManufacturerID*uint8[6] IPMI_OEMDefinedData*uint32 IANA*"

For non-timestamped OEM events, value of:
"*uint8[2] IPMI_RecordID*uint8 IPMI_RecordType*uint8[13] IPMI_OEMDefinedData*"

If the SEL timestamp has a value of 0xFFFFFFFF (which indicates an invalid or unspecified time value) or 0x00000000 through 0x20000000 (which indicates a time value that is relative BMC power-up and NOT 1970), a value of 0 should be set for CIM_LogRecord. MessageTimestamp.

4.5.2.7 CIM_LogRecord. RecordData

The following are examples of the Record.Data value.

System events:
for normal events, values like:
"*10.0.32*16 0*2*182 59 84 65*32 0*4*1*10*true*1*87*99*90*1*"
for shutdown events, values like:
"*0.0.65*48 0*2*182 99 84 65*65 0*4*32*0*true*111*3*255*255*1*"
for BSOD events, values like:
"*0.0.65*96 0*2*182 5 85 65*65 0*4*32*0*true*111*2*255*255*1*"

Timestamped OEM events, values like:
"*128 0*192*56 87 84 65*94 43 0*4 128 0 0 0 0*1*"

Non-Timestamped OEM events, values like:
"*256 0*253*56 87 84 65 0 1 0 4 128 0 0 0 0*"

While ordinary users of the CIM interface may not understand the specific data encoded into a RecordData property, a few users and OEM instrumentation and IPMI aware applications might know enough about IPMI to find this data useful. This is what the CIM model intends for this deliberately free-form (RecordData) property. See the IPMI specification, table 26-1, bytes 8-16 for details on what we expose in RecordData.

Aside from IPMI details, we also expose the CIM_Sensor.DeviceID key. This key value uniquely identifies the instance of CIM_Sensor (i.e. either NumericSensor or Sensor) that is associated with this particular SEL record.

4.5.2.8 CIM_LogRecord.Caption

EStr(<SEL Sensor.Type)>) || EStr(<SEL Generic or Sensor-Specific State Code>)

Procedure for EStr(<SEL Generic or Sensor Specific State Code>)

```
1809    #  SEL RecordType is byte 3 of SEL record (see Table 26-1)
1810    #  Only SEL RecordType == 02h is a “system event record”.
1811    #  Others are OEM defined, reserved, etc.  All further comments
1812    #  tend to assume SEL RecordType == 02h.
1813
1814    #  SEL Event Message Revision (EvM) is byte 10 of SEL Record
1815    #  (see Table 26-1).  IPMI v1.5 uses 04h.
1816    #  Future Direction:   Support IMPI v1.0 (03h) and/or IPMI v2.0 (05h?)
1817
1818    #  SEL Event Class is byte 13, bits [6:0] of SEL Record (see Table 26-1).
1819    #  The three possible values for EventClass are taken from
1820    #  Event/ReadingReadingTypeCodes as defined in Table 36-1.
1821    #  The three possible EventClass values are listed in  Table 23-6.
1822    #  The EventClass dictates how  to interpret the 3 subsequent bytes of
1823    #  Event Data found in each SEL Record’s bytes 14-16.  We call
1824    #  these three Event Data bytes EB1, EB2 and EB3.
1825
1826    #  SEL Event Direction is byte 13, bit [7] of each SEL
1827    #  Record.  We abbreviate the each SEL Record’s Event
1828    #  Direction as EDir.
1829
1830    #  SELSensorType is byte 11 of the SEL Record (see Table 26-1).
1831
1832    if <SEL RecordType> == 02h  and <EvM> == 04h
1833       if <SEL Event Class > == 01h  or
1834          (<SEL Event Class > >= 02h  and <SEL Event Class> <= 0Ch)
```

```
1835        if <SEL EDir> == 1b
1836          strEDir = “Assert: ”
1837        else  #  <SEL EDir> == 0b
1838          strEDir = “Deassert: “
1839        if <SEL EB1 [7:6]> == 11b
1840          #  Use SEL SensorType and EB2 value to index into Table 36-3
1841          #  to find a Sensor Specific Offset/State description string.  EB2
1842          #  is the Sensor-Specific Offset/State value..
1843          strOffsetOrStateDesc = strEDir + <EStr(<EB2 into Table36-3>)>
1844          return strOffsetOrStateDesc
1845        else if <SEL EB1 [6:5]> == 11b
1846          #  Use SEL SensorType and EB3  to index into Table 36-3 to find
1847          #  a Sensor Specific Offset/State description string.  EB3
1848          #  is the Sensor-Specific Offset/State value.
1849          strOffsetOrStateDesc = strEDir + <EStr(<EB3 into Table36-3>)>
1850          return strOffsetOrStateDesc
1851        else  #  We’ll have to use the Generic Offset/State description strings
1852          #  Use SEL EB1 [3:0] bits as a Generic Offset/State value.  Use
1853          #  this value, and the Event/ReadingTypeCode , to index into
1854          #  Table 36-1 to find a Generic Offset/State description string.
1855          strOffsetOrStateDesc = strEDir + <EStr(<EB1[3:0] into Table36-1>)>
1856          return strOffsetOrStateDesc
1857      else  #  This isn’t a SEL Reading Type code that the IPMI-to-CIM
1858            #   mapping supports.  Skip this SEL record.  It will not
1859            #   be mapped to an instance of the LogRecord class.  In other
1860            #   words, the IPMI-to-CIM mapping ignores OEM type SEL
```

Records.

4.5.2.9 CIM_LogRecord.Description

(string)(<CIM_Sensor.Name>) || ": " || (string)(EStr(CIM_Sensor.SensorType)) || " for " || EStr(<IPMI_SDR_EntityID>) || " " || (string)(<IPMI_SDR_EntityInstanceNumber>) || "; " EStr(<SEL Generic or Sensor-Specific State Code>)

See the procedure under CIM_LogRecord.Caption for a procedural description of the following term:

EStr(<SEL Generic or Sensor-Specific State Code>).

A SEL record's associated CIM_Sensor instance (whether that is a NumericSensor or a Sensor instance) can be found using the SEL Record's GeneratorID (bytes 8-9) and Sensor Number (byte 12). See Table 26-1 for details. This SEL data maps to the CIM_Sensor.DeviceID key value (as defined in the IPMI-to-CIM sensor mapping document). Once this Sensor key string is extracted from the SEL entry in question, the corresponding CIM_Sensor instance can be found.

Behind the scenes, the corresponding CIM_Sensor maps to a corresponding IPMI SDR record (or either type 1 or type 2). This indirectly corresponding SDR record must also be found in order to extract the IPMI_SDR_EntityID and IPMI_SDR_EntityInstanceNumber mentioned above.

We refer to the CIM_Sensor instance above because CIM_Sensor is a common base class for both NumericSensor and Sensor (as "trivially derived" for this IPMI-to-CIM mapping).

5 IPMI Commands Mapping

5.1 Introduction

The IPMI Commands Mapping allows the execution of a controlled subset of available IPMI commands using a CIM_CommandService class derived from CIM_Service.

5.2 Mapping Requirements

5.2.1 IPMI Commands Limitations

The IPMI Commands Mapping is the CIM object model to the IPMI interface mapping needed to issue and receive OEM "raw" messages to and from an embedded Baseboard Management Controller or Service Processor with IPMI compliant firmware. The class modeled in this profile is related to the IPMI BMC Messages as detailed in the IPMI Specification.

This model is only intended to map the execution of IPMI OEM commands. Execution of standard IPMI commands is NOT supported by this class. Standard commands are handled internally by the Sensor and Log Mapping implementations described in this document.

Also note that one instance of the CIM_CommandService class MUST be created for each IPMI capable microcontroller, in order to ensure that IPMI OEM commands destined for satellite controllers are routed correctly

5.3 CIM_CommandService Mapping

5.3.1 CIM_CommandService.Methods

5.3.1.1 Method: ExecuteOEMCommand

```
uint32 ExecuteOEMCommand(
   [IN] uint8 NetFN,
   [IN] uint8 CommandIdentifier,
   [IN] uint8 RequestPayload[],
   [IN ( false ), OUT] uint8 ResponseNetFN,
   [IN ( false ), OUT] uint8 ResponseCommandIdentifier,
   [IN ( false ), OUT] uint8 CompletionCode,
   [IN ( false ), OUT] uint8 ResponsePayload[])
```

MUST check for valid range of command request, as only OEM IPMI commands are supported by this method. The request is issued to the BMC only if successfully verified. Valid input parameter ranges for OEM commands are as follows:

1. NetFN in range 30h through 3Fh, as described in IPMI v1.5 specification, section 5.1, table 5.1
2. CommandIdentifier in range 00h through FDh., as described in IPMI v1.5 specification, section 5.0

3. When RequestPayload is assembled with NetFN, CommandIdentifier, and other request message fields, the total message length must conform to limits specified in IPMI V1.5 specification, Sections 6.13 and Appendix D.

The appropriate message bytes in the resulting response are assigned to the ResponseNetFN, ResponseCommandIdentifier, CompletionCode, and ResponsePayload[] output parameters . Note that message CompletionCode output parameter MUST support all valid values described in IPMI v1.5 specification, section 5.2, table 5.2, including device-specific OEM completion codes ranges 01h – 7eh.

Return value:
0 – Completed with no error
1 – Not Supported
2 – Unspecified Error
3 – Timeout
4 – Failed
5 – Invalid Parameter
6 – NetFN Out Of Range
7 – CommandIdentifier Out Of Range
8 – RequestPayload too long

5.3.2 CIM_CommandService.Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that listed in the Required section in Table 5 below MUST be implemented as defined in the subsection following this table unless the Notes column for each property specifies the defining reference.

Table 5 CIM_CommandService Class Properties

Required Properties	Notes
CreationClassName	**Key:**
Name	**Key:**
SystemCreationClassName	**Key:**
SystemName	**Key:**
IANA	**Key:**

Recommended Properties	Notes
Caption	
Description	
Started	Not Defined.
StartMode	Not Defined.
Status	Not Defined.
InstallDate	Not defined.

Recommended Properties	Notes

1942

1943 The following subsections describe procedures for generating values for various properties of
1944 CIM classes required in this profile.

1945 **5.3.2.1 CIM_ CommandService.SystemCreationClassName**

1946 Formulate as a string:

1947 <ComputerSystem.Name of ComputerSystem instance of the one associated BMC>

1948 This is the scoping system's CreationClassName.

1949 **5.3.2.2 CIM_ CommandService.SystemName**

1950 Name of the system that is hosting the CommandService

1951 **5.3.2.3 CIM_CommandService.CreationClassName**

1952 Fixed value of "CommandService"

1953 **5.3.2.4 CIM_ CommandService.Name**

1954 Name of IPMI controller to which commands will be routed, such as "Primary BMC", "OEM
1955 satellite controller 1", "OEM satellite controller 2", etc.

1956 **5.3.2.5 CIM_ CommandService.IANA**

1957 IANA ID for microcontroller

1958 **5.3.2.6 CIM_ CommandService. Caption**

1959 Fixed value of "IPMI Command Service"

1960 **5.3.2.7 CIM_ CommandService. Description**

1961 Fixed value of "IPMI Command Service"

1962

6 IPMI Power Control Mapping

6.1 Introduction

The IPMI Power Control Mapping extends the management capability of the IPMI to CIM mapping by adding the capability to manage the power state of the managed system and the BMC via IPMI represented as a CIM_PowerManagementService class.

The following class diagram describes the CIM classes involved in representing IPMI power control by the CIM_PowerManagementService class. The ComputerSystem classes are shown in this diagram to illustrate the way the CIM_PowerManagementService class is associated with overall class ecosystems. This is in preparation for interoperability with forthcoming profiles from the DMTF and other standards bodies.

Power Control Class Diagram

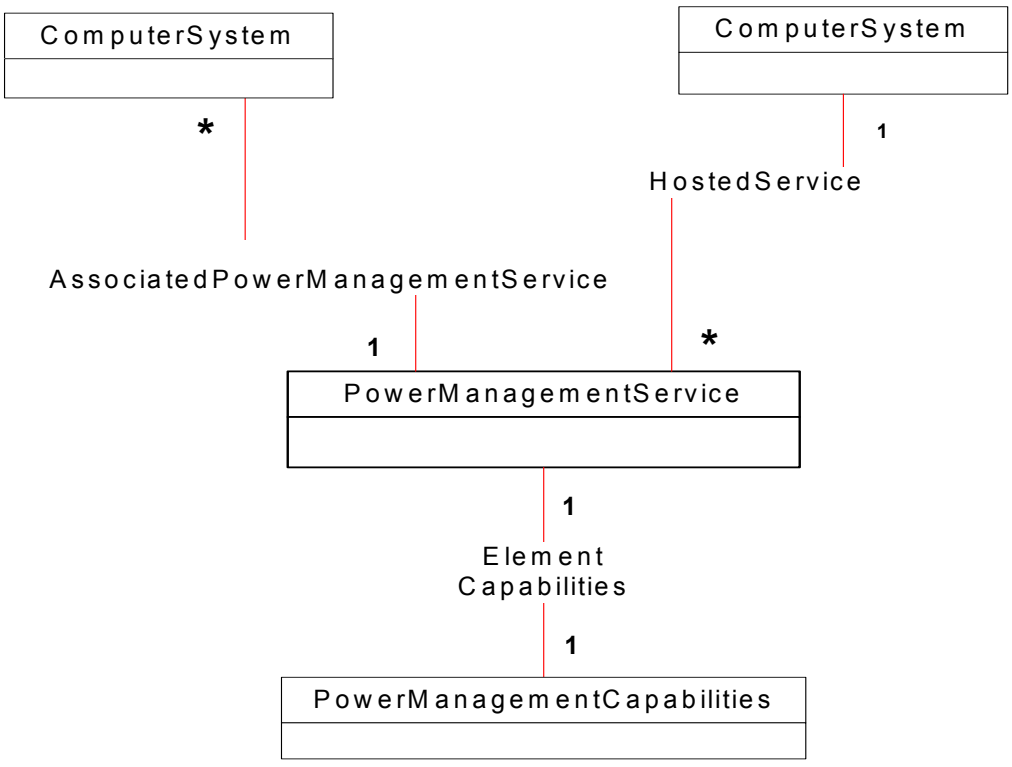


Figure 5 Class Diagram – IPMI Computer System Power Management

6.2 Instance Diagrams

The IPMI Computer System Power Management Mapping is the CIM object model to IPMI interface mapping needed to manage the managed or host system and the BMC power state.

The following instance diagram represents that instantiation of 2 CIM_PowerManagementService classes each responsible for representing the power control interface for either the BMC or the host system that the BMC is embedded in. The BMC and the host system are represented by instance of the CIM_ComputerSystem class.

Power Control Instance Diagram:

Monolithic Server with Service Processor

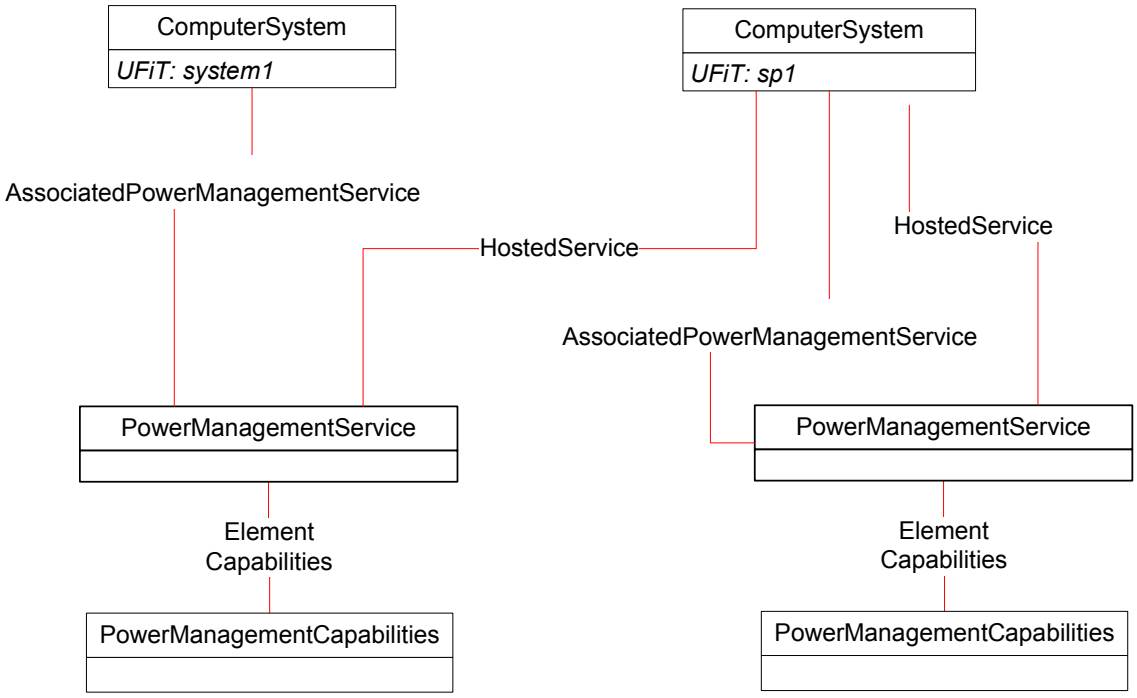


Figure 6 Instance Diagram – IPMI Computer System Power Management

6.3 Mapping Requirements

6.4 CIM_PowerManagementService Mapping

6.4.1 CIM_PowerManagementService Methods

6.4.1.1 Method: PowerManagementService.SetPowerState( )

```
uint32 SetPowerState( uint16 PowerState,
                      CIM_ManagedElement REF ManagedElement,
                      datetime Time
                    );
```

The CIM SetPowerState() method PowerState parameter is an enumerated value that corresponds to the IPMI Chassis Control command input parameter according to the following table:

CIM SetPowerState().PowerState Parameter	IPMI Chassis Command Parameter
Off-Hard	0h
On	01h
PowerCycle (Off-Hard)	02h
PowerCycle (Off-Soft)	03h
VendorDefined (for NMI)	04h
Off-Soft	05h

6.4.1.2 Method: PowerManagementService.StartService()

This method MUST return with a value of 1 or "Not Supported".

6.4.1.3 Method: PowerManagementServce.StopServce()

This method MUST return with a value of 1 or "Not Supported".

6.4.1.4 Method: PowerManagementService.RequestStateChange()

```
uint32 RequestStateChange (
   [IN] unit16 RequestedState,
   [IN] datetime TimeoutPeriod)
```

Return value:

0 – Completed with no error
1 – Not Supported
2 – Unspecified Error
3 – Timeout
4 – Failed
5 – Invalid Parameter
6..0x0FFF – DMTF_Reserved
0x1000..0x7FFF – Method_Reserved
0x8000.. – Vendor_Reserved

6.4.2 CIM_PowerManagementService.Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that listed in the Required section in Table 6 below MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

2021 **Table 6 CIM_PowerManagementService Class Properties**

Required Properties	Notes
CreationClassName	Key
Name	Key

2022

Recommended Properties	Notes
InstallDate	
StatusDescriptions	
Caption	
Description	
ElementName	

2023

2024 The following subsections describe procedures for generating values for various properties of
2025 CIM classes required in this profile.

2026 #### 6.4.2.1 CIM_PowerManagementService.CreationClassName

2027 Per the DMTF definition of the class property: The name of the class or the subclass used in the
2028 creation of an instance of the log for with this record is a part. This will typically be
2029 "CIM_PowerManagemetnService".

2030 Formulate as a string:

2031 <"CIM_PowerManagementService" or the name of the subclass created>

2032 #### 6.4.2.2 CIM_PowerManagementService.Name

2033 Fixed value of "IPMI Power Service"

2034 6.5 CIM_PowerManagementCapabilities Mapping

2035 6.5.1 CIM_PowerManagementCapabilities Methods

2036 None Defined.

2037 6.5.2 CIM_PowerManagementCapabilities Properties

2038 REQUIRED properties are properties that MUST be present. RECOMMENDED properties are
2039 properties that MAY be present according to t he guiding DMTF profiles.

2040 Properties that listed in the Required section in Table 7 below MUST be implemented as defined
2041 in the subsection following this table unless the Notes column for each property specifies the
2042 defining reference.

2043 **Table 7 CIM_PowerManagementCapabilities Class Properties**

Required Properties	Notes
PowerCapabilities	

2044

Recommended Properties	Notes
Caption	
Description	
ElementName	
OtherPowerCapabilitiesDescriptions	

2045 #### 6.5.2.1 CIM_PowerManagementCapabilities.PowerCapabilities

2046 Fixed value of 3 (for the enumeration "Power State Settable").

2047 ## 6.6 CIM_AssociatedPowerManagementService Mapping

2048 ### 6.6.1 CIM_AssociatedPowerManagementService Methods

2049 None Defined.

2050 ### 6.6.2 CIM_AssociatedPowerManagementService Properties

2051 REQUIRED properties are properties that MUST be present. RECOMMENDED properties are
2052 properties that MAY be present. For example, there may be properties that are only filled out in
2053 special configurations of the CIM Schema.

2054 Properties that are listed in the Required section in Table 8 below MUST be implemented as
2055 defined in the subsections following this table unless the Notes column for each property
2056 specifies the defining reference.

2057 **Table 8 CIM_AssociatedPowerManagementService Class Properties**

Required Properties	Notes
ServiceProvided	REF CIM_PowerManagementService {key, *}
UserOfService	REF ManagedElement {key, *}
PowerState	Values { "Other", "On", "Sleep - Light", "Sleep -Deep","Off - Hard", "Hibernate (Off - Soft)", "Off - Soft", "DMTF Reserved","Vendor Specific" },
OtherPowerState	
PowerOnTime	

2058

Recommended Properties	Notes
Caption	
Description	

2059 **6.6.2.1 CIM_AssociatedPowerManagementService.PowerState**

2060 **6.6.2.2 CIM_AssociatedPowerManagementService.OtherPowerState**

2061 **6.6.2.3 CIM_AssociatedPowerManagementService.PowerOnTime**

2062

7 IPMI User ID Management Mapping

7.1 Introduction

The IPMI User ID Management Mapping extends the management capability of the IPMI to CIM mapping by adding the capability to manage the User ID, passwords, and privileges of the BMC.

The following class diagram describes the CIM classes involved in representing IPMI User ID's, passwords, and privileges. The AdminDomain and ComputerSystem classes are shown in this diagram to illustrate the way the various classes involved in user id management are associated with overall class ecosystems. This is in preparation for interoperability with forthcoming profiles from the DMTF and other standards bodies.

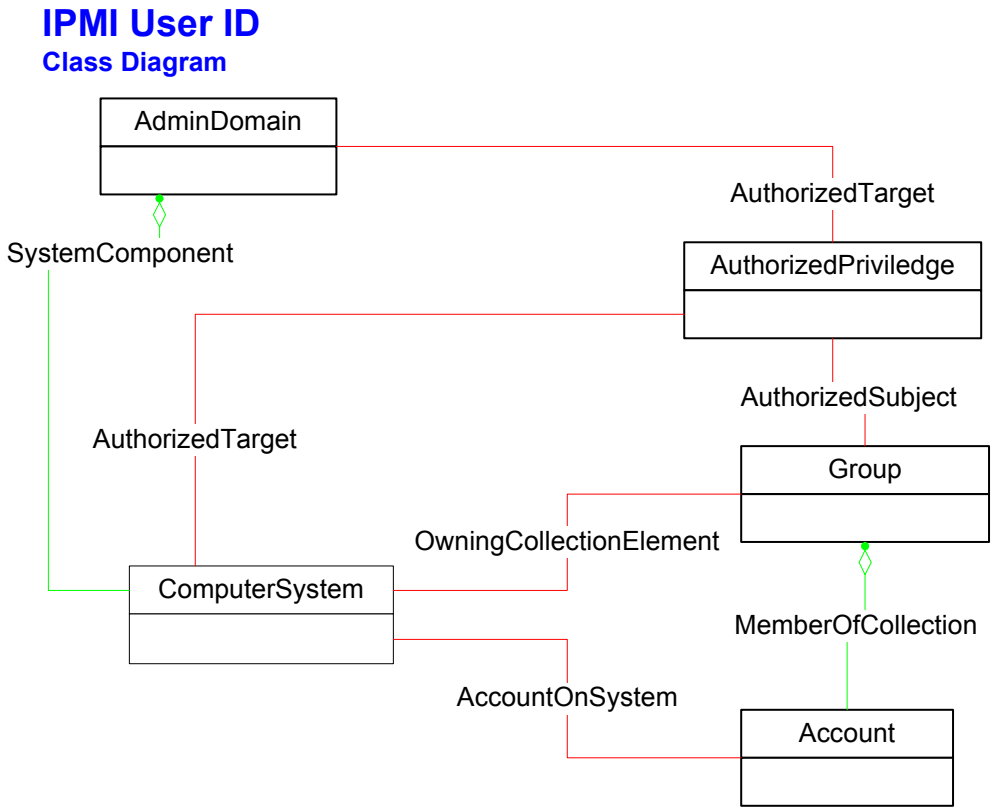


Figure 7 Class Diagram – IPMI User ID Management

7.2 Instance Diagrams

The IPMI User ID Management Mapping is the CIM object model to IPMI interface mapping needed to manage the BMC User ID's. The BMC and the host system are represented by instance of the CIM_ComputerSystem class.

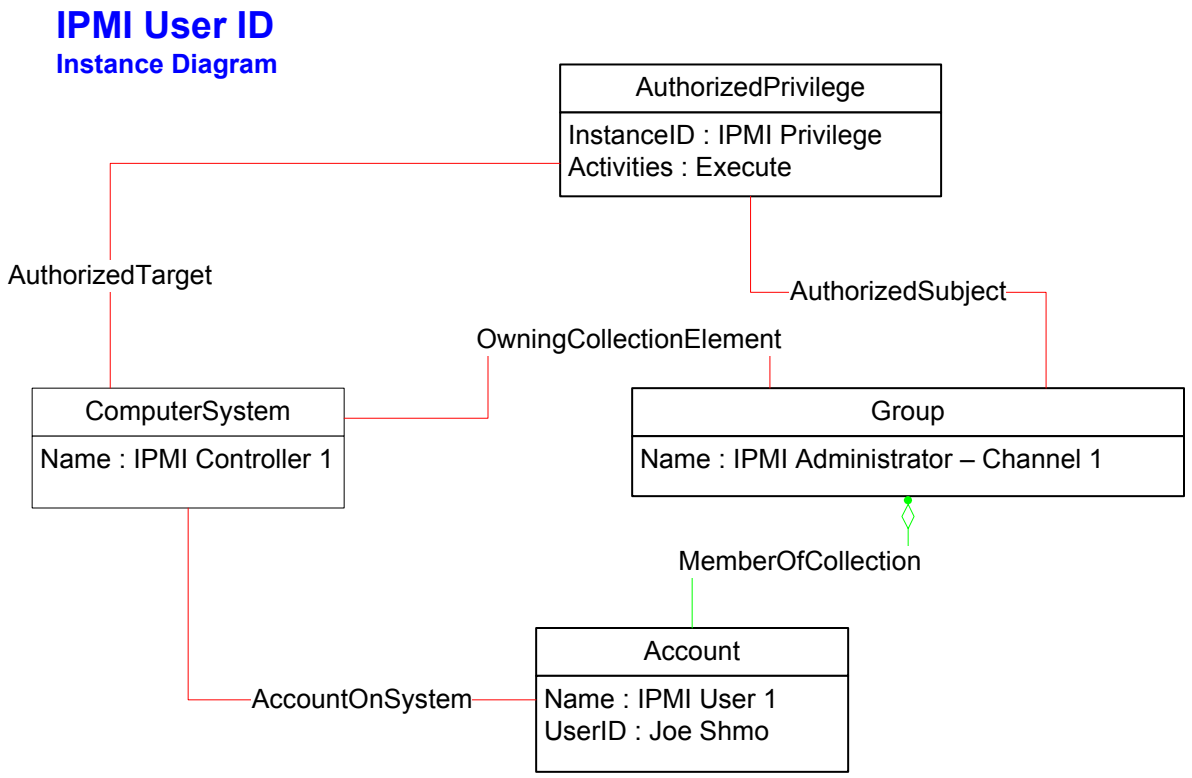


Figure 8 Instance Diagram – IPMI User ID Management

7.3 Mapping Requirements

7.3.1 Mapping IPMI User ID's

The IPMI User ID's are to be treated as slots that always exist irregardless of whether the username has been defined or whether the User ID is active. The implementation will instantiate the CIM_Account instance for the number of User ID's the IPMI implementation supports.

7.3.2 Mapping IPMI Access Privileges

Instances of CIM_Group are created for each of the possible privilege levels (eg Administrator, Operator, User, Callback, OEM, No Access) per Channel that an IPMI User ID may be granted. A specific CIM_Account instance may be a member of several groups, in which case the highest level access/execute privilege is in force. The access/execute privilege level in descending order is:

Group Privilege Level Order – Channel 1
Administrator – Channel 1
No-Access – Channel 1
Operator – Channel 1

Group Privilege Level Order – Channel 1
User – Channel 1
Call-Back – Channel 1
OEM – Channel 1

For every channel available in the implementation, a set of instances of CIM_Group corresponding to the possible proviledge levels for that channel will need to be instantiated.

Example for an implementation with 2 channels: The following 12 instances of CIM_Group are required:

Groups for Channel 1	Groups for Channel2
Administrator – Channel 1	Administrator – Channel 2
No-Access – Channel 1	No-Access – Channel 2
Operator – Channel 1	Operator – Channel 2
User – Channel 1	User – Channel 2
Call-Back – Channel 1	Call-Back – Channel 2
OEM – Channel 1	OEM – Channel 2

7.3.3 Modeling CIM_AuthorizedPrivilege

The CIM_AuthorizedPrivilege is modeled as granting "execute" authorization to the various group it is assocated with. The implementation MUST instantiate one instance of CIM_AuthorizedPriviledge and associate each IPMI group with it.

7.4 CIM_Account Mapping

7.4.1 CIM_Account Methods

None Defined.

7.4.2 CIM_Account Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that listed in the Required section in Table 9 below MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

2112 **Table 9 CIM_Account Class Properties**

Required Properties	Notes
SystemCreationClassName	Key
SystemName	Key
CreationClassName	Key
Name	Key
UserID	
UserPassword	
OperationalStatus	
StatusDescriptions	

2113

Recommended Properties	Notes
ObjectClass	
Descriptions	
Host	
LocalityName	
OrganizationalName	
OU	
SeeAlso	
UserCertificate	
InstallDate	
HealthState	
Caption	
Description	
ElementName	

2114

2115 The following subsections describe procedures for generating values for various properties of
2116 CIM classes required in this profile.

2117 #### 7.4.2.1 CIM_Account.SystemCreationClassName

2118 Per the DMTF definition of the class property: The scoping System's CreationClassName. In all
2119 cases this will be the class “CIM_ComputerSystem”.

2120 Formulate as a string:

2121 <”CIM_ComputerSystem”>

2122 #### 7.4.2.2 CIM_Account.SystemName

2123 Formulate as a string:

<ComputerSystem.Name of ComputerSystem instance of the one associated BMC>

The System instance that represents the associated management controller has a Name property. This be found and then copied in here. Since there is but one instance of a management controller for most IPMI-compliant devices/systems, this is often quite easy to do.

7.4.2.3 CIM_Account.CreationClassName

Per the DMTF definition of the class property: The name of the class or the subclass used in the creation of an instance of the log for with this record is a part. This will typically be "CIM_Account".

Formulate as a string:

<"CIM_Account" or the name of the subclass created>

7.4.2.4 CIM_Account.Name

IPMI User ID slot number.

Formulate as a string:

"IPMI User" || <IPMI User ID>

Where the IPMI User ID is a number from 1 to N where N is the number of User ID's supported by the IPMI implementation. An instance of CIM_Account MUST be instantiated for every possible used, including those cases where the UserName is Null.

7.4.2.5 CIM_Account.UserID

IPMI User Name.

Formulate as a string:

<IPMI User Name>

The IPMI User Name is retrieved from the IPMI command GetUsername().

7.4.2.6 CIM_Account.UserPassword

IPMI Password.

This is always Null for read operations.

7.4.2.7 CIM_Account.OperationalStatus

2151 #### 7.4.2.8 CIM_Account.StatusDescriptions

2152 ## 7.5 CIM_Group Mapping

2153 ### 7.5.1 CIM_Group Methods

2154 None Defined.

2155 ### 7.5.2 CIM_Group Properties

2156 REQUIRED properties are properties that MUST be present. RECOMMENDED properties are
2157 properties that MAY be present according to t he guiding DMTF profiles.

2158 Properties that listed in the Required section in Table 10 below MUST be implemented as
2159 defined in the subsection following this table unless the Notes column for each property specifies
2160 the defining reference.

2161 **Table 10 CIM_Group Class Properties**

Required Properties	Notes
CreationClassName	Key
Name	Key

2162

Recommended Properties	Notes
BusinessCategory	
CommonName	
Caption	
Description	
ElementName	

2163 #### 7.5.2.1 CIM_Group.CreationClassName

2164 Per the DMTF definition of the class property: The name of the class or the subclass used in the
2165 creation of an instance of the log for with this record is a part. This will typically be
2166 “CIM_Group”.

2167 Formulate as a string:

2168 <”CIM_Group” or the name of the subclass created>

2169 #### 7.5.2.2 CIM_Group.Name

2170 Instances of CIM_Group MUST be created for each of the following IPMI user level privileges.

2171 Formulate as a string the following IPMI user level privileges:

2172 IPMI Administrator

IPMI Operator

IPMI User

IPMI Callback

IPMI OEM

IPMI No Access

7.6 CIM_AuthorizedPrivilege Mapping

7.6.1 CIM_AuthorizedPrivilege Methods

None Defined.

7.6.2 CIM_AuthorizedPrivilege Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that listed in the Required section in Table 11 below MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

Table 11 CIM_AuthorizedPrivilege Class Properties

Required Properties	Notes
InstanceID	Key
PrivilegeGranted	Boolean
Activities	
ElementName	

Recommended Properties	Notes
ActivityQualifiers	
QualifierFormats	
Caption	
Description	

7.6.2.1 CIM_AuthorizedPrivilege.InstanceID

Formulate as a string:

"IPMI Privilege"

7.6.2.2 CIM_AuthorizedPrvilege.PrivilegeGranted

Formulate as a Boolean:

Yes

7.6.2.3 CIM_AuthorizedPrivilege.Activities

Formulate as an integer enumeration:

7 (Enumeration = “Execute”)

7.6.2.4 CIM_AuthorizedPrivilege.ElementName

Formulate as a string:

“IPMI Execute Privilege”

8 IPMI BMC Mapping

8.1 Introduction

The IPMI BMC Mapping extends the management capability of the IPMI to CIM mapping by adding the capability to model and manage the BMC as a Manageability Access Point (MAP). Also modeled is the managed system that the BMC is embedded in.

The following class diagram describes the CIM classes involved in representing BMC admin domain, the BMC itself and the managed system the BMC is embedded in. This is in preparation for interoperability with forthcoming profiles from the DMTF and other standards bodies.

The AdminDomain class is for supporting DMTF SMASH CLP addressability and is consistent with the modeling approach.

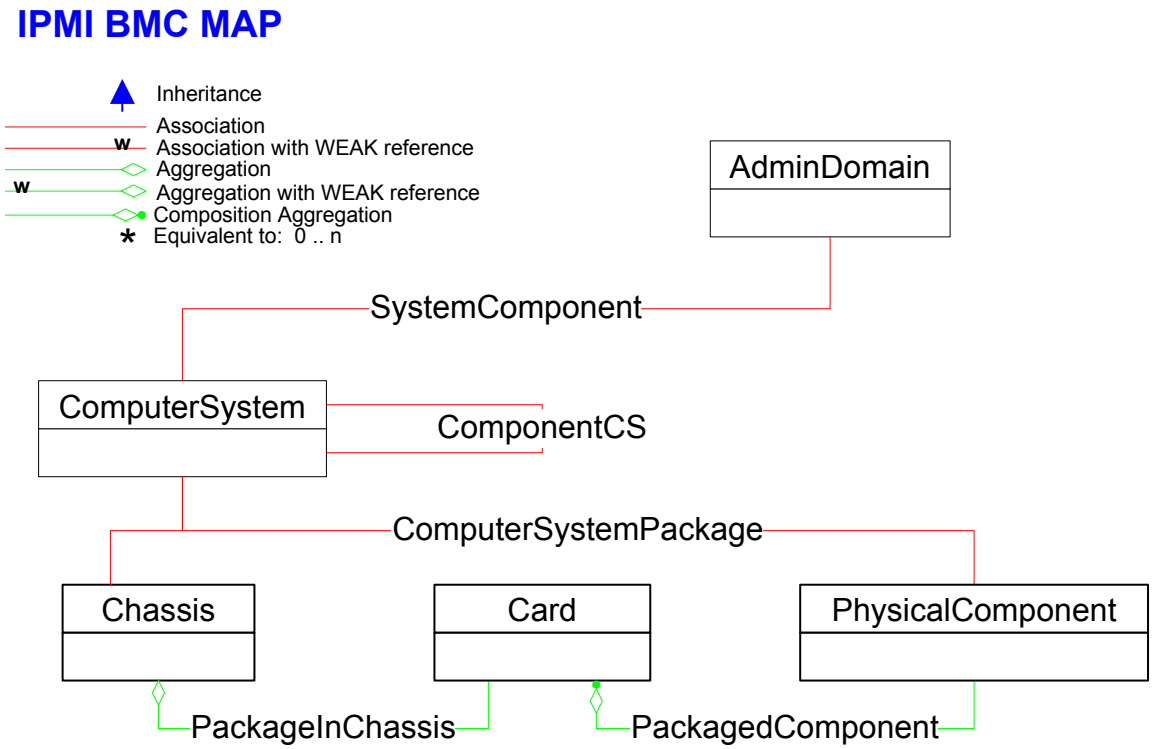


Figure 9 Class Diagram – IPMI BMC MAP Management

8.2 Instance Diagrams

The IPMI BMC MAP Management mapping is the CIM object model to IPMI interface mapping needed to represent and manage BMC.

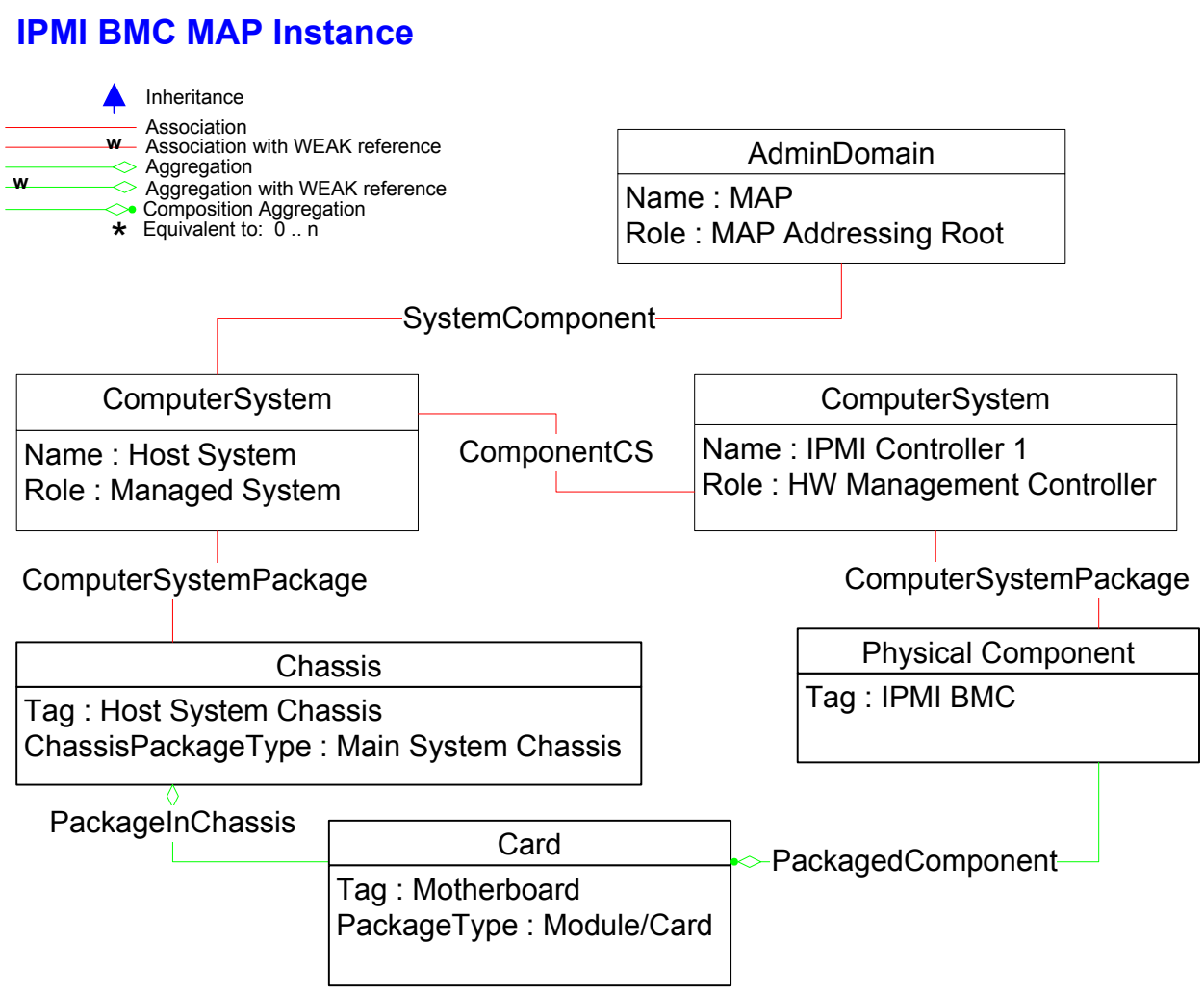


Figure 10 Instance Diagram – IPMI BMC MAP Management

8.3 Mapping Requirements

The implementation MUST instantiate a single instance of CIM_AdminDomain in the implementation namespace and MUST instantiate two (2) instances of CIM_ComputerSystem. The first instance of CIM_ComputerSystem is to represent the BMC and the second instance is to represent the host or managed system.

Physical classes such as CIM_Chassis and CIM_Card SHOULD be instantiated if the implementation has FRU and/or other host system inventory data, including the host system name.

8.4 CIM_AdminDomain Mapping

8.4.1 CIM_AdminDomain Methods

8.4.1.1 Method: AdminDomain.RequestStateChange()

```
uint32 RequestStateChange (
   [IN] unit16 RequestedState,
   [IN] datetime TimeoutPeriod)
```

Return value:

0 – Completed with no error
1 – Not Supported
2 – Unspecified Error
3 – Timeout
4 – Failed
5 – Invalid Parameter
6..0x0FFF – DMTF_Reserved
0x1000..0x7FFF – Method_Reserved
0x8000.. – Vendor_Reserved

8.4.2 CIM_AdminDomain.Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that are listed in the Required section in Table 6 below MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

Table 12 CIM_AdminDomain Class Properties

Required Properties	Notes
CreationClassName	Key
Name	Key

Recommended Properties	Notes
NameFormat	
PrimaryOwnerName	
PrimaryOwnerContact	
Roles	
EnabledState	
OtherEnabledState	
RequestedState	
EnabledDefault	

Recommended Properties	Notes
TimeOfLastStateChange	
InstallDate	
OperationalStatus	
StatusDescriptions	
HealthState	
Caption	
Description	
ElementName	

The following subsections describe procedures for generating values for various properties of CIM classes required in this profile.

8.4.2.1 CIM_AdminDomain.CreationClassName

Per the DMTF definition of the class property: The name of the class or the subclass used in the creation of an instance of the log for with this record is a part. This will typically be "CIM_AdminDomain".

Formulate as a string:

<"CIM_AdminDomain" or the name of the subclass created>

8.4.2.2 CIM_AdminDomain.Name

Fixed value of "MAP"

8.5 CIM_ComputerSystem Mapping

8.5.1 CIM_ComputerSystem Methods

8.5.1.1 Method: ComputerSystem.RequestStateChange()

```
uint32 RequestStateChange (
   [IN] unit16 RequestedState,
   [IN] datetime TimeoutPeriod)
```

Return value:

0 – Completed with no error
1 – Not Supported
2 – Unspecified Error
3 – Timeout
4 – Failed
5 – Invalid Parameter

2277 6..0x0FFF – DMTF_Reserved
2278 0x1000..0x7FFF – Method_Reserved
2279 0x8000.. – Vendor_Reserved

2280 ### 8.5.2 CIM_ComputerSystem Properties

2281 REQUIRED properties are properties that MUST be present. RECOMMENDED properties are
2282 properties that MAY be present according to t he guiding DMTF profiles.

2283 Properties that listed in the Required section in Table 7 below MUST be implemented as defined
2284 in the subsection following this table unless the Notes column for each property specifies the
2285 defining reference.

2286 **Table 13 CIM_ComputerSystem Class Properties**

Required Properties	Notes
Name	KEY
CreationClassName	KEY
Dedicated	
Roles	
EnabledState	
HealthState	
OperationalStatus	

2287

Recommended Properties	Notes
NameFormat	
OtherIdentifyingInfo	
IdentifyingDescriptions	
OtherDedcatedDescriptions	
ResetCapability	
PrimaryOwnerName	
PrimaryOwnerContact	
RequestedState	
OtherEnabledState	
EnabledDefault	
TimeOfLastStateChange	
InstallDate	
StatusDescriptions	
Caption	
Description	
ElementName	

2288

The following sections may describe the values for two instances of CIM_ComputerSystem. The values for the instance of CIM_ComputerSystem that represent the BMC are labeled with a "**BMC:**" and the value definitions for the instance that represent the managed system are labeled "**Host:**".

8.5.2.1 CIM_ComputerSystem.Name

BMC:

Concatenation of "IPMI Controller" and BMC device ID (Get Device ID response byte 2)

Formulate as a string:

<"IPMI Controller" || string (Get Device ID response byte 2)>

Example Value: "IPMI Controller 32"

Host:

The goal for the value of the Name property for the instance of CIM_ComputerSystem representing the IPMI host or managed system is to provide a correlatable identifier that can be used by applications to correlate representations from different manageability access points. At present, due to various versions of IPMI being available in the field and a general lack of consensus in the industry as to what correlatable identifier should be predominant, the value formulation presented here is an attempt to define a formulation that can be generated across various situations.

The general format of the Name property shall be:

< (NameValue) || ":" ||

string (GUID as retrieved from Get System GUID IPMI Command)>

Where NameValue is generated in the following fashion:

IF the implementation resides in-band in the OS resident on the system containing the BMC, the implementation should use the value of the Name property in the instance of CIM_ComputerSystem generated by OS instrumentation and representing the managed system for NameValue.

Else IF the MultiRecord Area is available in the system FRU and the Management Access Record (Type 0x03) Sub-Record Type (0x02) System Name string is populated, use this value for NameValue.

Else IF the IPMI interface supports the IPMI 2.0 command Get System Info, use the value returned in the System Name parameter for NameValue.

Else use the following value for NameValue:

< "Unknown" || "." || "IPMI BMC Device ID" || "." ||

2322 string (Device ID as retrieved from the Get Device ID IPMI command)>

2323 #### 8.5.2.2 CIM_ComputerSystem.CreationClassName

2324 Per the DMTF definition of the class property: The name of the class or the subclass used in the
2325 creation of an instance of the log for with this record is a part. This will typically be
2326 "CIM_ComputerSystem".

2327 Formulate as a string:

2328 <"CIM_ComputerSystem" or the name of the subclass created>

2329 #### 8.5.2.3 CIM_ComputerSystem.Dedicated

2330 **BMC:**

2331 For the instance of CIM_ComputerSystem representing the IPMI BMC:

2332 This is a fixed value: 14 (enumerated as "Management Controller")

2333 **Host:**

2334 The value of the Dedicated property for the managed system is left up to the implementation to
2335 select the appropriate value from the enumeration value/value maps defined in the
2336 CIM_ComputerSystem MOF. However, the when implementation is representing a general
2337 purpose server it should use the value of:

2338 0 (enumerated as "Not Dedicated")

2339 #### 8.5.2.4 CIM_ComputerSystem.Roles

2340 This property is writeable and may be changed by the user/application.

2341 **BMC:**

2342 For the instance of CIM_ComputerSystem representing the IPMI BMC:

2343 The initial value: "Hardware Management Controller"

2344 **Host:**

2345 For the instance of CIM_ComputerSystem representing the managed system:

2346 The initial value: "Managed System"

8.5.2.5 CIM_ComputerSystem.EnabledState

8.5.2.6 CIM_ComputerSystem.HealthState

8.5.2.7 CIM_ComputerSystem.OperationalStatus

8.5.2.8 CIM_ComputerSystem.NameFormat

Format of the Name string generated by this mapping:

This is a fixed value of "Other"

8.5.2.9 CIM_ComputerSystem.ElementName

BMC:

Concatenation of "IPMI Controller" and BMC device ID (Get Device ID response byte 2)

Formulate as a string:

<"IPMI Controller" || string (Get Device ID response byte 2)>

Example Value: "IPMI Controller 32"

Host:

The goal for the value of the Name property for the instance of CIM_ComputerSystem representing the IPMI host or managed system is to provide a correlatable identifier that can be used by applications to correlate representations from different manageability access points. At present, due to various versions of IPMI being available in the field and a general lack of consensus in the industry as to what correlatable identifier should be predominant, the value formulation presented here is an attempt to define a formulation that can be generated across various situations.

The general format of the Name property shall be:

< (NameValue) || ":" ||

string (GUID as retrieved from Get System GUID IPMI Command)>

Where NameValue is generated in the following fashion:

IF the implementation resides in-band in the OS resident on the system containing the BMC, the implementation should use the value of the Name property in the instance of CIM_ComputerSystem generated by OS instrumentation and representing the managed system for NameValue.

Else IF the MultiRecord Area is available in the system FRU and the Management Access Record (Type 0x03) Sub-Record Type (0x02) System Name string is populated, use this value for NameValue.

Else IF the IPMI interface supports the IPMI 2.0 command Get System Info, use the value returned in the System Name parameter for NameValue.

Else use the following value for NameValue:

< “Unknown” || “.” || “IPMI BMC Device ID” || “.” ||

string (Device ID as retrieved from the Get Device ID IPMI command)>

8.5.2.10 CIM_ComputerSystem.IdentifyingDescriptions

Fixed value as: “Manufacturer ID-Product ID”

8.5.2.11 CIM_ComputerSystem.OtherIdentifyingInfo

Concatenation of BMC Manufacturer ID (Get Device ID response byte 8-10) and BMC Product ID(Get Device ID response byte 11-12)

Formulate as a string:

< string (Get Device ID response byte 8-10) ||

string (Get Device ID response byte 11-12) >

Example:

11102-12345

8.5.2.12 CIM_ComputerSystem.Caption

Concatenation of “BMC ” and its device ID (Get Device ID response byte 2)

Formulate as a string:

< “BMC “ || string (Get Device ID response byte 2) >

Example:

“BMC 32”

8.5.2.13 CIM_ComputerSystem.Description

Concatenation of BMC device ID (Get Device ID response byte 2), IPMI version (Get Device ID response byte 6), BMC firmware revision (Get Device ID response byte 4-5), BMC auxiliary firmware revision (Get Device ID response byte 13-16)

Formulate as a string:

< “BMC “ || string (Get Device ID response byte 2) || “ – IPMI v” ||
string (Get Device ID response byte 6) || “ – fw v” ||
string (Get Device ID response byte 4-5) || “.” ||
string (Get Device ID response byte 13-16) >

Exmaple:

“BMC 32 - IPMI v1.5 - fw v0.30.0.0.0.0”

9 IPMI Versioning Mapping

9.1 Introduction

The IPMI Versioning Mapping extends the management capability of the IPMI to CIM mapping by adding the capability to model the version of the mapping being implemented

The following class diagram describes the CIM classes involved in representing the IPMI to CIM mapping version being implemented. The use of the CIM_RegisteredProfile class to represent the version of the IPMI to CIM "Mapping" is not completely aligned with the description of the class. However, due to the lack of another class suitable for representing the Mapping version information, the mapping will reuse the CIM_RegisteredProfile class. The implementation should instantiate the instance of CIM_RegisteredProfile in the implementation namespace so as to avoid having to support cross namespace associations.

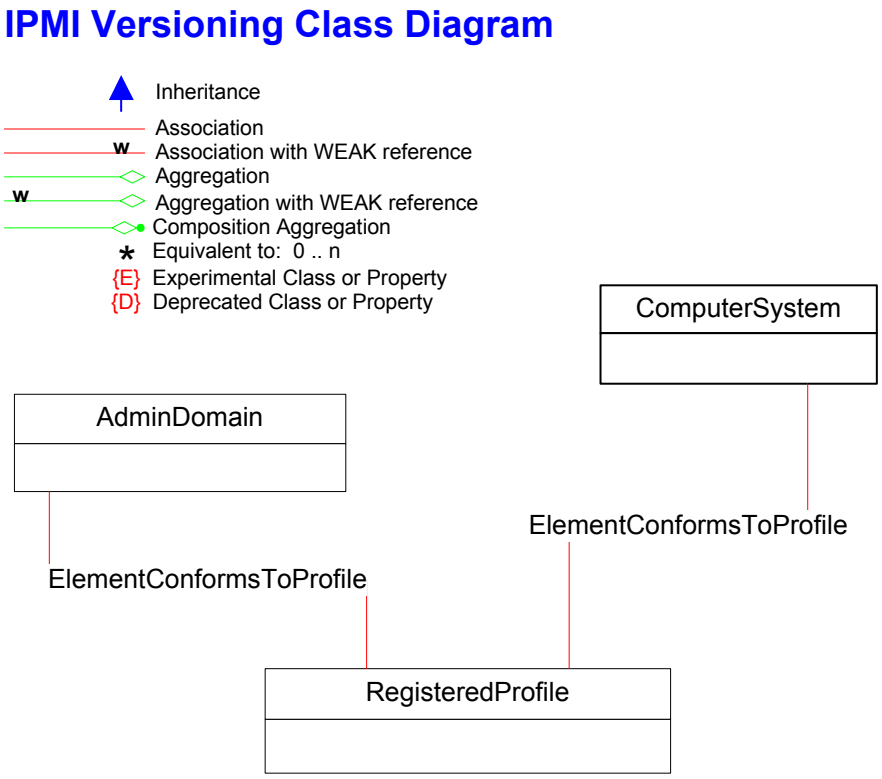


Figure 11 Class Diagram – IPMI Versioning

9.2 Instance Diagrams

The IPMI Versioning mapping is the CIM object model representation of the version of the IPMI to CIM mapping implemented.

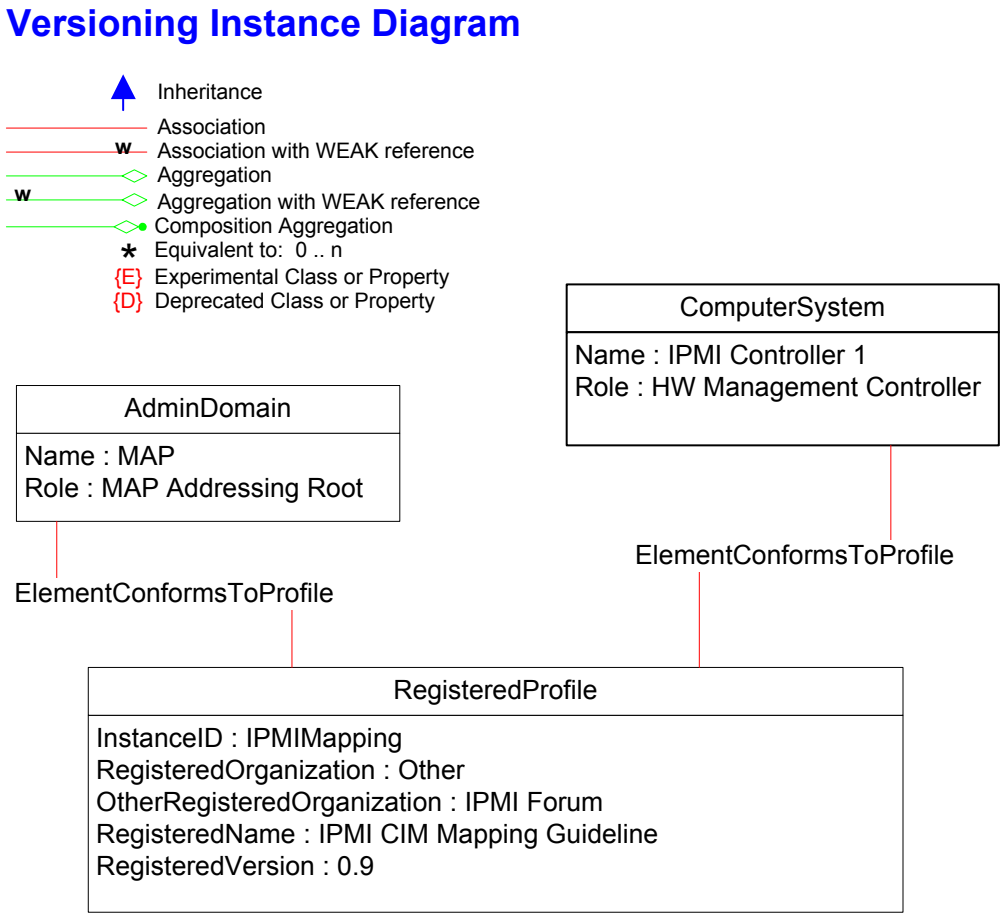


Figure 12 Instance Diagram – IPMI Versioning

9.3 Mapping Requirements

9.4 CIM_RegisteredProfile Mapping

9.4.1 CIM_RegisteredProfile Methods

None.

9.4.2 CIM_RegisteredProfile Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that are listed in the Required section in Table 6 below MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

2440 **Table 14 CIM_RegisteredProfile Class Properties**

Required Properties	Notes
InstanceID	Key
RegisteredOrganization	
OtherRegisteredOrganization	
ReqisteredName	
RegisteredVersion	

2441

Recommended Properties	Notes
AdvertiseTypes	
AdvertiseTypeDescriptions	
Caption	
Description	
ElementName	

2442

2443 The following subsections describe procedures for generating values for various properties of
2444 CIM classes required in this profile.

2445 #### 9.4.2.1 CIM_RegisteredProfile.InstanceID

2446 Formulate as string:

2447 "IPMI : Mapping"

2448 #### 9.4.2.2 CIM_RegisteredProfile.RegisteredOrganization

2449 Formulate as uint16:

2450 1 – (Other)

2451 #### 9.4.2.3 CIM_RegisteredProfile.OtherRegisteredOrganization

2452 Formulate as string:

2453 "IPMI Forum"

2454 #### 9.4.2.4 CIM_RegisteredProfile.RegisteredName

2455 Formulate as string:

2456 "IPMI CIM Mapping Guideline"

2457 #### 9.4.2.5 CIM_RegisteredProfile.RegisteredVersion

2458 Formulate as string:

2459 “Version” || <version of this IPMI CIM Mapping Guideline>

10 IPMI Entity Mapping

10.1 Introduction

The IPMI Entity Mapping extends the management capability of the IPMI to CIM mapping by adding the capability to model the physical entities and corresponding logical devices discoverable via the IPMI interface. FRU information, where available, will be represented in the physical class being used to represent the physical entity. Logical devices are instantiated where a corresponding physical class exists and can be identified from the entity type.

The following class diagram describes the CIM classes involved in representing the physical entities defined by the IPMI interface.

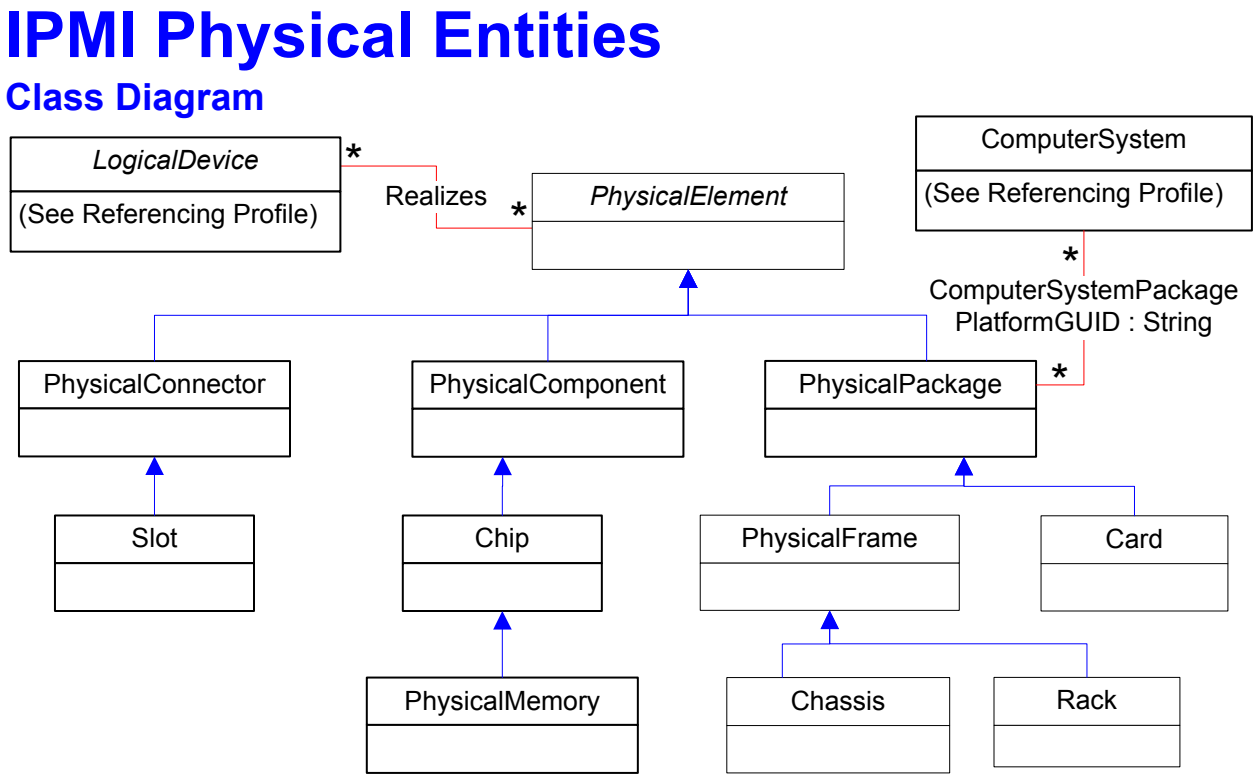


Figure 13 Class Diagram – IPMI Physical Entities

The following class diagram describes the CIM associations involved in associating physical classes according to entity in entity information from the IPMi interface.

Physical Entity Associations

Association Toplogy Diagram

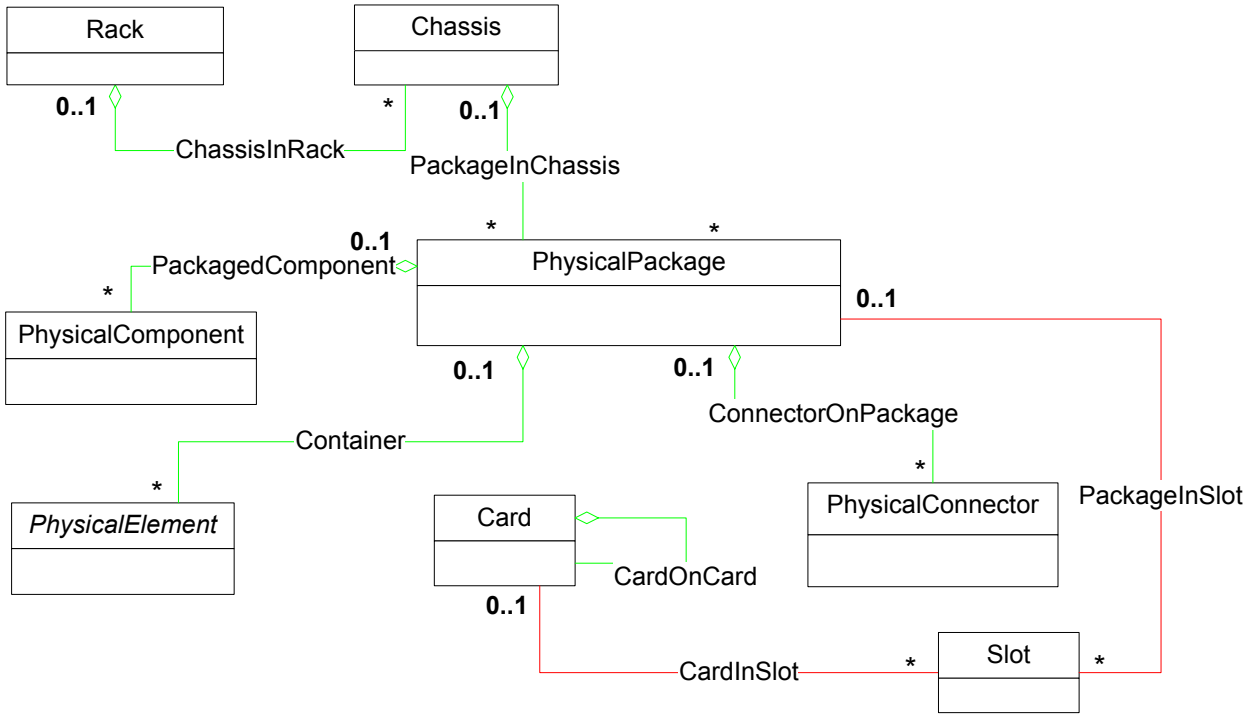


Figure 14 Class Diagram – IPMI Physical Associations

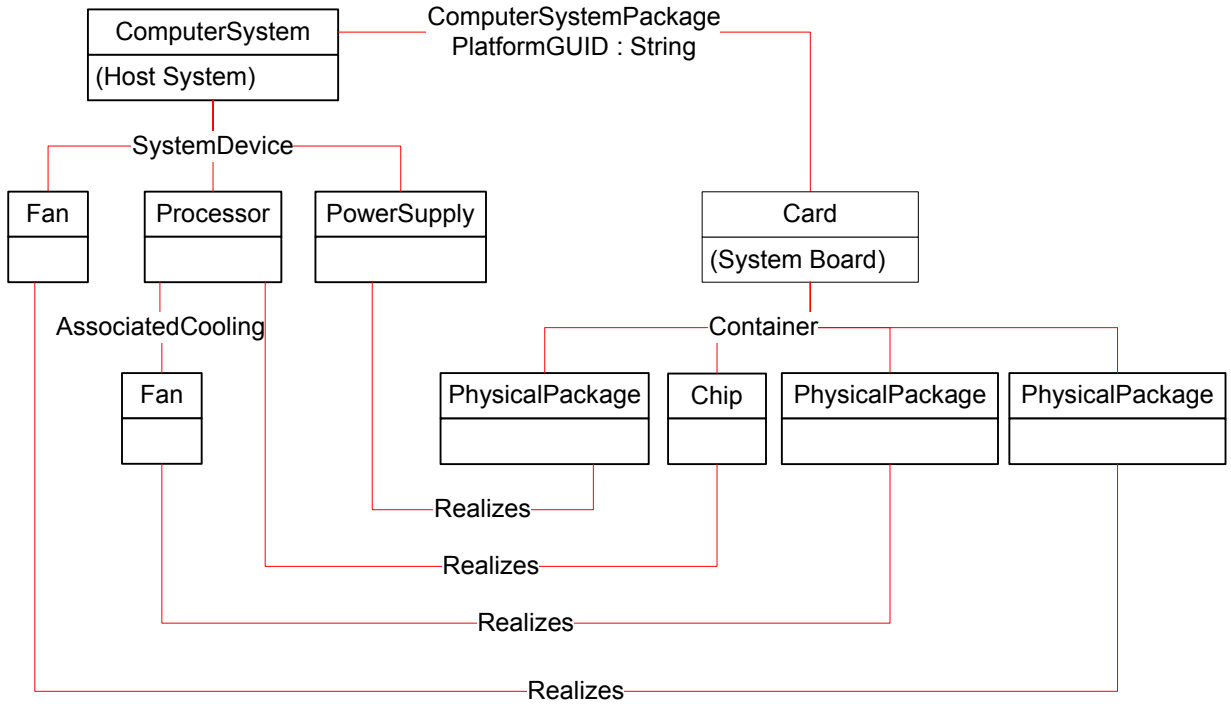
10.2 Instance Diagrams

10.2.1 Example: System Board as Physical Parent

The following diagram describes a possible instantiation of physical and logical CIM classes generated from entity information available via the IPMI interface. In this example, the IPMI implementation has used the IPMI Entity "System Board" as the parent or top level container for components in the system.

IPMI Entities

Instance Diagram (System Board as physical parent)



2482

2483 **Figure 15 Instance Diagram – IPMI Entities (System Board Parent)**

2484 ### 10.2.2 Example: Chassis as Physical Parent

2485 The following diagram describes a possible instantiation of physical and logical CIM classes
2486 generated from entity information available via the IPMI interface. In this example, the IPMI
2487 implementation has used the IPMI Entity "System Chassis" as the parent or top level container
2488 for components in the system.

IPMI Entities

Instance Diagram (Chassis as physical parent)

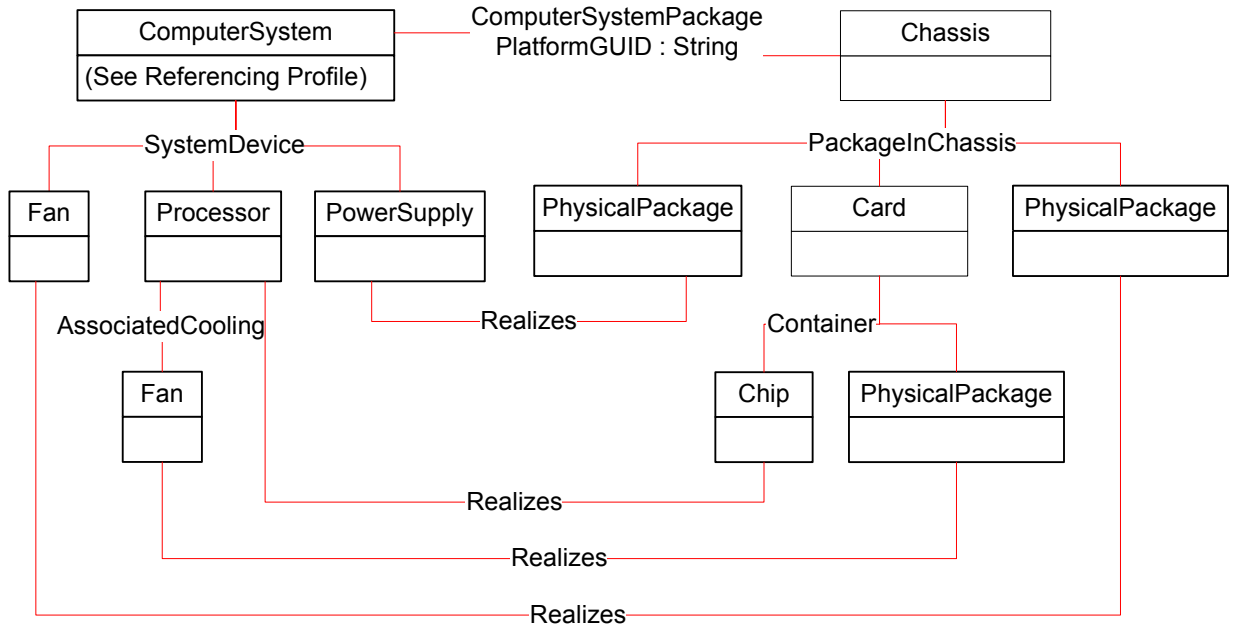


Figure 16 Instance Diagram – IPMI Entities (Chassis Parent)

10.3 Mapping Requirements

Mapping IPMI entities and containers to CIM classes involves instantiating physical CIM classes to model hierarchical relationships and to provide data structures for FRU information. In many cases, CIM logical device classes also need to be instantiated to provide actual manageability of the components being modeled in addition to providing representations that are meaningful to consumers. Examples are the modeling of a Fan entity using both CIM_PhysicalPackage and CIM_Fan classes for the case where both physical and logical are instantiated and the modeling of a Disk Drive Bay where only the CIM_PhysicalPackage physical class is used.

10.3.1 Mapping Entity ID's to CIM Classes

The following table defines the mapping from the IPMI Entity IDs (Table 37.13 IPMI 1.5 Specification) to physical and logical CIM classes. Note that some entity ID types do not have corresponding logical device types in CIM. Additionally, the IPMI entity id's representing software are mapped to CIM_SoftwareIdentity, which is not a physical class.

IPMI Entity	CIM Physical Class	CIM Logical Device
Processor (03h)	CIM_Chip	CIM_Processor

IPMI Entity	CIM Physical Class	CIM Logical Device
Disk or disk bay (04h)	CIM_PhysicalPackage	CIM_DiskDrive
Peripheral bay (05h)	CIM_PhysicalPackage	
System management module (06h)	CIM_PhysicalPackage	CIM_ComputerSystem (BMC)
System board (07h)	CIM_Card	CIM_ComputerSystem (Main System)
Memory module (08h)	CIM_PhysicalPackage	CIM_Memory
Processor module (09h)	CIM_PhysicalPackage	CIM_Processor
Powey supply (0Ah)	CIM_PhysicalPackage	CIM_PowerSupply
Add-in card (0Bh)	CIM_Card	CIM_ComputerSystem
Front panel board (0Ch)	CIM_Card	
Back panel board (0Dh)	CIM_Card	
Power system board (0Eh)	CIM_Card	CIM_PowerSupply
Drive backplane (0Fh)	CIM_Card	
System internal expansion board (10h)	CIM_Card	CIM_ComputerSystem
Other system board (11h)	CIM_Card	
Processor board (12h)	CIM_Card	CIM_Processor
Power unit/power domain (13h)	CIM_PhysicalPackage	CIM_PowerSupply
Power module/DC-to-DC converter (14h)	CIM_PhysicalPackage	CIM_PowerSupply
Power management/power distribution board (15h)	CIM_Card	CIM_PowerSupply
Chassis back panel board (16h)	CIM_Card	
System chassis (17h)	CIM_Chassis	CIM_ComputerSystem (Managed System)
Sub-chassis (18h)	CIM_Chassis	
Other chassis board (19h)	CIM_Card	
Disk drive bay (1Ah)	CIM_PhysicalPackage	
Peripheral bay (1BH)	CIM_PhysicalPackage	
Device bay (1Ch)	CIM_PhysicalPackage	
Fan/cooling unit (1Eh)	CIM_PhysicalPackage	CIM_Fan
Cable/interconnect (1Fh)	CIM_PhysicalPackage	
Memory device (20h)	CIM_PhysicalPackage	CIM_Memory
System management software (21h)	CIM_SoftwareIdentity	
System firmware (22h)	CIM_SoftwareIdentity	
Operating system (23h)	CIM_SoftwareIdentity	
System bus (24h)	CIM_PhysicalPackage	
Group (25h)		
Remote management card (26h)	CIM_Card	CIM_ComputerSystem
External environment (27h)		
Battery (28h)	CIM_PhysicalPackage	CIM_Battery
Processing blade (29h)	CIM_PhysicalPackage	CIM_ComputerSystem
Connectivity switch (2Ah)	CIM_PhysicalPackage	CIM_ComputerSystem
Processor/memory module (2Bh)	CIM_PhysicalPackage	CIM_ComputerSystem
I/O module (2Ch)	CIM_PhysicalPackage	CIM_ComputerSystem

IPMI Entity	CIM Physical Class	CIM Logical Device
Processor/IO module (2Dh)	CIM_PhysicalPackage	CIM_ComputerSystem
Management controller firmware (2Eh)	CIM_SoftwareIdentity	
IPMI Channel (2Fh)		CIM_ServiceAccessPort

2507 **Table 15 IPMI Entity ID to CIM Class Mapping**

2508

2509 ### 10.3.2 Entity Instantiation Algorithm

2510 The following algorithm is presented as the method for determining when to instantiate entities
2511 as CIM classes. Use the Table 15 IPMI Entity ID to CIM Class Mapping to determine which
2512 classes to instatiate according to entity ID.

2513 An "Entity" in IPMI is identified by the following pieces of information: The "Entity ID" which
2514 identifies the type of the Entity the "Entity Instance", which is a number that uniquely identifies
2515 an instance of the entity - either relative to the overall system, or relative to a particular
2516 management controller in the system.

2517 In the context of IPMI, the term "Entity" or "Entities" in this section shall be used as a shorthand
2518 notation to refer to the full unique identification of a particular Entity. (That is, the combination
2519 of Entity ID, Entity Instance, and, if the Entity Instance is device-relative, the identifying
2520 information for the management controller associated that together uniquely identify an Entity or
2521 Entities.)

2522 IPMI supports a type of Sensor Data Records called Entity Association Records. These records
2523 provided a way of representing a hierarchical relationship among Entities by using linking a set
2524 of 'contained' Entities under a single 'container' Entity.

2525 A given Entity is only allowed only be contained under a single, present, Entity Association
2526 hierarchy at a time. I.e. it can't appear in more than one existing Entity Association where the
2527 top-level container entity is present. However, it's possible that the SDRs could hold records for
2528 two or more Entity Association hierarchies, where one hierarchy is present, and the others are
2529 'absent' because the container entity is absent (as indicated by a presence sensor or presence bit
2530 associated with that container entity) . This is because some implementations may elect to
2531 implement a set of SDRs that is a superset that covers multiple system configurations, and use
2532 the Entity Association record mechanism as a way to organizing which sensors/Entities are
2533 present for a given system configuration.

2534 Because of this, it is necessary to track which hierarchy a given Entity appears under. That way,
2535 it allows an Entity to be 'do not instantiate' under one hierarchy, but can still exist to be
2536 instantiated under another. This requires the following rule: if an Entity is listed under *any* Entity
2537 Association hierarchy, then it must be part of a 'present' hierarchy in order to be instantiated.

2538 The following figure illustrates the case where a set of Entity Association records (left hand side
2539 of figure) should be viewed as the equivalent of two separate Entity Association hierarchies
2540 because two different container Entities are sharing common contained Entities. Note that this

requires one association to be ‘present’ and the other ‘absent’. In this case, this is accomplished via an Entity Presence sensor that is associated with each ‘top’ container Entity.

If the container Entity does not have any sensors associated with it, then the association is assumed to be ‘present’.

If the container Entity has one or more sensors associated with it, but all sensors are ‘scanning disabled’ or are inaccessible, then the association should be ignored. I.e. software should act as if that association was not even listed in the SDRs.

If the container Entity has one or more ‘scanning enabled’ associated with it, and none of those sensors are a presence sensor or presence bit that indicates that the container Entity is absent, then the association is assumed to be ‘present’ (valid).

If the container Entity has a presence sensor or bit associated with it that indicates the container Entity is absent, then the overall association is said to be ‘*explicitly absent*’ and the container Entity and any contained Entities and lower level associations should not be instantiated because of being listed under that association. (Note that the same Entities may still become instantiated because they’re listed under a different Entity association or association hierarchy that *is* present.)

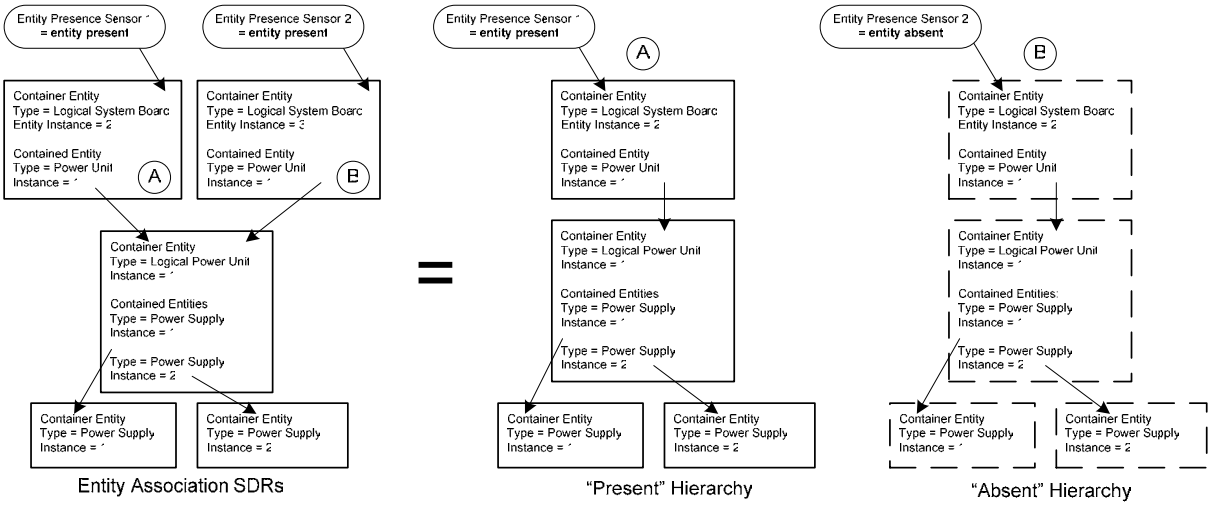


Figure 17 “Present” and “Absent” Entity Association Hierarchies

1. First, deterimine which Type 01h and Type 02h SDRs can be ignored because their management controller is not present. Once these SDRs have been identified, they do not need to be re-examined in any further steps of this Entity instantiation process.

    - The BMC is always considered present and therefore does not require a device locator record.

    - If the management controller associated with the sensor is not listed via a Management Controller Device Locator Record (except for the BMC), assume

any sensors associated with that controller are not present. This relationship can be identified by examining the fields for Device Slave Address and Channel Number listed in the Management Controller Device Locator record. These same values are used in the Type 01h and Type 02h sensor data records.

- For each management controller listed, issue a *Get Device ID* command to verify that the controller is accessible. If the controller is not accessible, ignore any Type 01h and Type 02h SDRs associated with sensors for that controller.

2. Next, create a list of Entities that exist under Entity Associations, tagging each entry in the list with which association the Entity was found under. An Entity could possibly appear more than once in the list.

   - Start by finding the topmost 'root' container Entities. These are Entities that are not themselves listed as being contained Entities in any other Entity Association record. (The top of every Entity hierarchy must have a unique container Entity. Thus, the id information for the container Entity can be used as a way to identify a particular hierarchy)

   - Next, for each hiearchy, add the contained Entities to the list under that hierarchy, and then see if any of the contained Entities are container entities for any other entity association records. If so, add their contained entities to the list under the hierarchy and repeat the process until no more container entities are found under the hierarchy.

   - You now should have a list of the different Entity hierarchies (Entity Associations) and the Entities contained in each hierarchy. Note that at this point, a given Entity could appear in more than one hierarchy.

3. Next, identify which Entities in the list should not be instantiated because they are contained under the hierarchy of a Device-relative Entity Association record where the management controller associated with the top of that heirarchy is not present.

   - For each Device-related Entity Association record, check to see whether the management controller for the record is present and accessible. If the management controller is not present and accessible, mark the contained Entities in the list as 'do not instantiate'.

   - Check to see if any of the 'do not instantiate' Entities from the previous step are themselves container entities for either a Device-relative or regular (type 08h) Entity Association record under the hierarchy. If so, mark any subsequent contained Entities in that hierarchy as 'do not instantiate'. Repeat until no more container entities are found that are listed in the 'do not instantiate' list.

4. Next, identify which Entities should not be instantiated because they are contained under an Entity Assocation (either Device-related or regular [type 08h]) where the container Entity is 'not present' because an Entity Presence sensorindicates that the container Entity is not present.

- Check the Type 01h and Type 02h records and see if there is an 'Entity Presence' sensor associated with the container Entity. If so, access the sensor to see if the status indicates 'Entity Absent'. If it does, mark the entity as 'do not instantiate' under the hierarchy.

- If the Entity Presence sensor is disabled (not scanning) or inaccessible, assume the Entity is not present and also mark the entity as 'do not instantiate' under the hierarchy.

At this point, any Entities in the hierarchy lists that are **not** marked as 'do not instantiate' are available to be instantiated. So you should go through the hierarchies and create two lists of Entities: a 'available for instantiation' list of all the Entities the are are part of a 'present' hierachy, and a 'do not instantiate' list, which is the list Entities that were found in an Entity Association hierarchy and are not on the 'avaiable for instantiation' list.

5. Identify whether additional Entities should be instantiated because they're listed in Type 01h, 02h, or 03h SDRs.

   - For each Type 01h, Type 02h, or Type 03h SDR (that isn't to be ignored because of step '1') see if the Entity associated with the sensor is on the 'do not instantiate' list. If so, go on to the next SDR.

   - Otherwise, check to see if the sensor is of a type that includes a "Presence" bit that indicates whether the Entity associated with the sensor is present or not. (This only applies to Type 01h or Type 02h SDRs that support readable sensors). If the bit indicates the Entity is absent, add that Entity to the 'do not instantiate' list and remove it from 'available from instantiation' list if it's there . Otherwise, add the Entity to the 'available for instantiation' list.

6. Certain types of sensors imply an Entity that can be instantiated under CIM. For example, Fan and Processor sensors imply the existence of Fan and Processor entities. Identify Entities that should be instantiated based on the sensor type.

   - For each Type 01h, Type 02h, or Type 03h SDR (that isn't to be ignored because of step '1') see if the Entity associated with the sensor is on the 'do not instantiate' list. If so, go on to the next SDR.

   - If the Sensor Type (Sensor/Event Type) in the SDR does not match the Entity ID given in the SDR, then a separate Entity based on the sensor type should be added to the 'available for instantiation' list, provided that the Sensor/Event Type is one of the types in the following table. Since there is not an explicit Entity Instance number available in this case, an Entity Instance number must be synthesized. The Entity Instance number should recored as the combination of Sensor Number (SDR type 1|2 table byte 8), Owner LUN (SDR type 1|2 table byte 7), Owner ID (SDR type 1|2 table byte 6) so that the instance ID can ultimately be formulated as a CIM_Sensor.DeviceID [See section 3.4.2.4, CIM_Sensor.DeviceID].

Table 16 "Physical" Sensor Types to Entity Mapping

Sensor Type	Sensor Type Code	Entity
Fan	04h	Fan (29)
Physical Security (Chassis Intrusion)	05h	System Chassis (23)
Processor	07h	Processor (3)
Power Supply (also used for power converters [e.g. DC-to-DC converters] and VRMs [voltage regulator modules]).	08h	Power Supply (10)
Power Unit	09h	Power Unit (19)
Cooling Device	0Ah	Cooling Device (29)
Button / Switch	14h	Sub-Chassis (18h)
Module / Board	15h	Other Chassis Board (19)
Microcontroller / Coprocessor	16h	Processor (3)
Add-in Card	17h	Add-In Card (11)
Chassis	18h	System Chassis (23)
Chip Set	19h	Processor (3)
Other FRU	1Ah	???
Cable / Interconnect	1Bh	Cable/Interconnect (31)
Terminator	1Ch	Cable/Interconnect (31)
Slot / Connector	21h	Cable/Interconnect (31)
Watchdog 2	23h	CIM_WatchDog
Monitor ASIC / IC	26h	CIM_Chip
LAN	27h	Remote Management Communication Device (38)
Battery	29h	Battery (28)
Session Audit	2Ah	???

7. Add Entities to instantiate because they're listed in FRU Device Locator records.

   - For each FRU Device Location record, check to see if the Entity is already listed in the 'available for instantiation' list. If not, add the Entity to the 'available for instantiation' list if the corresponding FRU Device is accessible.

8. Add Entities to instantiate because they're listed in Management Controller Device Locator records.

   - Primary FRU Devices are FRU devices that are implemented as FRU Device 000b on LUN 00b of a management controller. These devices do not have FRU Device Locator records. Instead, the presence of the FRU Device and information on the Entity associated with the FRU Device is provided by the Management Controller Device Locator record associated with the controller. For each Management Device Locator recored, check to see if the Entity is already listed in the 'available for instantiation' list. If not, add the Entity to the 'available for instantiation' list if the corresponding FRU Device is accessible.

9. Add Entities to instantiate because they're implied by FRU Data. Refer to the following table.

Table 17 FRU - Entity Instantiation

FRU Device is:	Enitity Type is:	FRU Device Contains:	FRU Data is for:
BMC Primary FRU	n/a	Chassis Info Area + Product Info	Assume Chassis Info is for overall

Device (FRU Device 000b on BMC LUN 00b)		Area	system chassis and Product Info is for overall computer system as a product, and that the Chassis Info is for instance 1 of the system chassis (Entity Type 11h) unless that Entity has already been explicitly identified by a FRU Device Locator or Management Controller Device Locator record. In that case, instantiate a unique Entity for a different instance of 'system chassis'.
		Product Info Area but no Chassis Info Area	Assume Product Info is for overall System UNLESS another FRU device is found that contains *both* a Chassis Info and a Product Info area and is associated with instance 1 of the system chassis (Entity Type 11h)
		Chassis Info Area but no Product Info Area	Assume Chassis Info is for overall System UNLESS another FRU device is found that contains *both* a Chassis Info and a Product Info area and is associated with instance 1 of the system chassis (Entity Type 11h).
		Board Info Area	Assume Board Info is for instance 1 of the main system board (Entity Type 7h) unless that Entity has been explicitly identified by a FRU Device Locator or Management Controller Device Locator record. In that case, instantiate a unique Entity for an 'other system board (11h)'.
Primary FRU Device on Satellite controller (Device 000b on LUN 00b)	Specified by Management Controller Device Locator Record	Board Info Area, Chassis Info Area, or Product Info area	FRU data is associated with type of Entity given by the Entity ID information in the Management Controller Device Locator record. If Chassis info is present, & there is no other Chassis info is present, assume info is for overall system chassis and Product Info, if also present, is for overall computer system as a product, and that the Chassis Info is for instance 1 of the system chassis (Entity Type 11h) unless that Entity has already been explicitly identified by a FRU Device Locator or Management Controller Device Locator record. In that case, instantiate a unique Entity for a different instance of 'system chassis'.
	Unspecified	Chassis Info Area + Product Info Area	If BMC Primary FRU Device does not already have Chassis Info and Product Info Areas, and a FRU area Assume Chassis Info is for overall system chassis and Product Info is for overall computer system as a product, and that the Chassis Info is for instance 1 of the system chassis (Entity Type 11h) unless that Entity has already been explicitly identified by a FRU Device Locator or

			Management Controller Device Locator record. Otherwise, instantiate a unique Entity for a different instance of 'system chassis' associated with the management controller.
		Product Info Area but no Chassis Info Area	Assume Product Info is for overall System UNLESS there is Product Info in the BMC Primary FRU Device, or another FRU device is found that contains *both* a Chassis Info and a Product Info area and is associated with instance 1 of the system chassis (Entity Type 11h)
		Chassis Info Area but no Product Info Area	Assume Chassis Info is for overall System UNLESS there is Chassis Info in the BMC Primary FRU Device, or another FRU device is found that contains *both* a Chassis Info and a Product Info area and is associated with instance 1 of the system chassis (Entity Type 11h)
		Board Info Area	Assume Board Info is for instance 1 of the main system board (Entity Type 7h) unless that Entity has been explicitly identified by a FRU Device Locator or Management Controller Device Locator record. In that case, instantiate a unique Entity for an 'other system board (11h)'.
Identified by FRU Device Locator Record	Specified by FRU Device Locator Record	Board Info Area, Chassis Info Area, or Product Info area	FRU data is associated with type of Entity given by the Entity ID information in the FRU Device Locator record. Any combination of Product, Chassis, and Board Info areas may be present. Appropriate CIM "Package" classes should be instantiated for each type of record. The Entity itself should be added to the 'available for instantiation' if not already present.
	Unspecified	Chassis Info Area + Product Info Area	If Chassis Info area is present, and a Chassis Info area is *not* provided in the Primary FRU Device of the BMC, assume Chassis Info is for overall system chassis and Product Info is for overall computer system as a product. Assume Chassis Info is for instance 1 of the system chassis (Entity Type 11h) unless that Entity has already been explicitly identified by a FRU Device Locator or Management Controller Device Locator record. In that case, instantiate a unique Entity for a different instance of 'system chassis'.
		Product Info Area but no Chassis Info Area	Assume Product Info is for overall System UNLESS another FRU device is found that contains *both* a Chassis Info and a Product Info area.
		Chassis Info Area but no Product Info Area	Assume Chassis Info is for overall System UNLESS another FRU device is found that contains *both* a Chassis Info and a Product Info area and is

			either associated with instance 1 of the system chassis (Entity Type 11h), or is in the Primary FRU Device of the BMC.
		Board Info Area	Board Info is for FRU that the management controller is located on.

10.3.3 Entity Instance Correlation

Instrumentation MUST keep track of the IPMI Entity Instance identifiers and correlate during SDR discovery so as to only produce one set of CIM instances representing a particular IPMI entity.

10.4 CIM_PhysicalPackage Mapping

10.4.1 CIM_PhysicalPackage Methods

None.

10.4.2 CIM_PhysicalPackage Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Several property vales are dependent on the availability of associated FRU information. The Product Info Area as defined in Platform Management FRU Information Storage Definition v1.1 is mapped for this class.

Properties that are listed in the Required section in the following table MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

Table 18 CIM_PhysicalPackage Class Properties

Required Properties	Notes
Tag	Key
CreationClassName	Key
PackageType	
VendorEquipmentType	
HealthState	
OperationalStatus	
Description	
ElementName	

Recommended Properties	Notes
VendorCompatibilityString(E)	

Recommended Properties	Notes
PoweredOn	
CanBeFRUed	
Manufacturer	If FRU available See 10.4.2.4
Model	If FRU available See 10.4.2.5
SKU	
OtherPackageType	
RemovalConditions	
ManufactureDate	
ActiveCooling	
Version	
SerialNumber	If FRU available See 10.4.2.7
PartNumber	If FRU available See 10.4.2.6
Height	
Depth	
Width	
Weight	
IdentifyingDescriptions	
OtherIdentifyingInfo	
UserTracking	
StatusDescriptions	
InstallDate	
Name	If FRU available See 10.4.2.4
Caption	

2682

2683 The following subsections describe procedures for generating values for various properties of
2684 CIM classes required in this profile.

2685 **10.4.2.1 CIM_PhysicalPackage.Tag**

2686 An arbitrary string that uniquely identifies the Physical Element and serves as the Element ' s
2687 key.

2688 Formulate as string:

2689 (string)<Entity ID from SDR> || (string)<Entity Instance from SDR>

2690 || (string)<Sensor Number from SDR>

2691 **10.4.2.2 CIM_PhysicalPackage.CreationClassName**

2692 Per the DMTF definition of the class property: The name of the class or the subclass used in the
2693 creation of an instance. This will typically be "CIM_PhysicalPackage".

Formulate as a string:

<”CIM_PhysicalPackage” or the name of the subclass created>

10.4.2.3 CIM_PhysicalPackage.PackageType

Values for CIM_PhysicalPackage.PackageType are mapped to values of Entity ID from the SDR.

Formulate as integer:

if Entity ID = 04h (disk or disk bay)

CIM_PhysicalPackage.PackageType = 15 (Storage Media Package)

elseif EntityID = 05h (peripherial bay)

CIM_PhysicalPackage.PackageType = 5 (Container/Frame Slot)

elseif EntityID = 06h (system management module)

CIM_PhysicalPackage.PackageType = 1 (Other)

CIM_PhysicalPackage.OtherPackageType = “System Management Module”

elseif EntityID = 08h (memory module)

CIM_PhysicalPackage.PackageType = 13 (Memory)

elseif EntityID = 09h (processor module)

CIM_PhysicalPackage.PackageType = 12 (Processor)

elseif EntityID = 0Ah (power supply)

CIM_PhysicalPackage.PackageType = 6 (Power Supply)

elseif EntityID = 13h (power unit/power domain)

CIM_PhysicalPackage.PackageType = 1 (Other)

CIM_PhysicalPackage.OtherPackageType = “Power unit/Power domain”

elseif EntityID = 14h (power module/DC-to-DC converter)

CIM_PhysicalPackage.PackageType = 1 (Other)

CIM_PhysicalPackage.OtherPackageType = “Power module/DC-to-DC converter”

elseif EntityID = 1Ah (disk drive bay)

2720 CIM_PhysicalPackage.PackageType = 1 (Other)

2721 CIM_PhysicalPackage.OtherPackageType = “Disk drive bay”

2722 elseif EntityID = 1Bh (peripheral bay)

2723 CIM_PhysicalPackage.PackageType = 1 (Other)

2724 CIM_PhysicalPackage.OtherPackageType = “Peripheral bay”

2725 elseif EntityID = 1Ch (device bay)

2726 CIM_PhysicalPackage.PackageType = 1 (Other)

2727 CIM_PhysicalPackage.OtherPackageType = “Device bay”

2728 elseif EntityID = 1Eh (fan/cooling unit)

2729 CIM_PhysicalPackage.PackageType = 7 (Fan)

2730 elseif EntityID = 1Fh (cable/interconnect)

2731 CIM_PhysicalPackage.PackageType = 1 (Other)

2732 CIM_PhysicalPackage.OtherPackageType = “Cable/Interconnect”

2733 Elseif EntityID = 20h (memory device)

2734 CIM_PhysicalPackage.PackageType = 13 (Memory)

2735 Elseif EntityID = 24h (system bus)

2736 CIM_PhysicalPackage.PackageType = 4 (Cross Connect/Backplane)

2737 elseif EntityID = 28h (battery)

2738 CIM_PhysicalPackage.PackageType = 1 (Other)

2739 CIM_PhysicalPackage.OtherPackageType = “Battery”

2740 elseif EntityID = 29h (processing blade)

2741 CIM_PhysicalPackage.PackageType = 1 (Other)

2742 CIM_PhysicalPackage.OtherPackageType = “Processing Blade”

2743 elseif EntityID = 2Ah (connectivity switch)

2744 CIM_PhysicalPackage.PackageType = 1 (Other)

2745 CIM_PhysicalPackage.OtherPackageType = “Connectivty Switch”

elseif EntityID = 2Bh (processor/memory module)

CIM_PhysicalPackage.PackageType = 1 (Other)

CIM_PhysicalPackage.OtherPackageType = “Processor/Memory Module”

elseif EntityID = 2Ch (I/O module)

CIM_PhysicalPackage.PackageType = 1 (Other)

CIM_PhysicalPackage.OtherPackageType = “I/O Module”

elseif EntityID = 2Dh (processor/IO module)

CIM_PhysicalPackage.PackageType = 1 (Other)

CIM_PhysicalPackage.OtherPackageType = “Processor/IO Module”

10.4.2.4 CIM_PhysicalPackage.Manufacturer

Formulate as a string:

(string) associated FRU 1.1 Product Info Area – Manufacturer Name field

10.4.2.5 CIM_PhysicalPackage.Model

Formulate as a string:

(string) associated FRU 1.1 Product Info Area – Product Part/Model Number field

10.4.2.6 CIM_PhysicalPackage.PartNumber

Formulate as a string:

(string) associated FRU 1.1 Product Info Area – Product Part/Model Number field

10.4.2.7 CIM_PhysicalPackage.SerialNumber

Formulate as a string:

(string) associated FRU 1.1 Product Info Area – Product Serial Number field

10.4.2.8 CIM_PhysicalPackage.Name

Formulate as a string:

(string) associated FRU 1.1 Product Info Area – Product Name field

10.4.2.9 CIM_PhysicalPackage.HealthState

10.4.2.10 CIM_PhysicalPackage.OperationalStatus

10.4.2.11 CIM_PhysicalPackage.Description

Formulate as string:

(string)< CIM_PhysicalPackage.PackageType or
CIM_PhysicalPackage.OtherPackageType > || “(” || (string)<Entity ID from SDR> || “:” ||
(string)<Entity Instance from SDR>

10.4.2.12 CIM_PhysicalPackage.ElementName

Formulate as string:

(string)< CIM_PhysicalPackage.PackageType or
CIM_PhysicalPackage.OtherPackageType > || “(” || (string)<Entity ID from SDR> || “:” ||
(string)<Entity Instance from SDR>

10.5 CIM_Card Mapping

10.5.1 CIM_Card Methods

None.

10.5.2 CIM_Card Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Several property vales are dependent on the availability of associated FRU information. The Board Info Area as defined in Platform Management FRU Information Storage Definition v1.1 is mapped for this class.

Properties that are listed in the Required section in the following table MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

2795 **Table 19 CIM_Card Class Properties**

Required Properties	Notes
Tag	Key
CreationClassName	Key
HostingBoard	
PackageType	
OtherPackageType	
HealthState	
OperationalStatus	
Description	
ElementName	

2796

Recommended Properties	Notes
Manufacturer	If FRU available See 10.5.2.5
Model	
SKU	
VendorEquipmentType	
VendorCompatibilityString(E)	
PoweredOn	
CanBeFRUed	
SlotLayout	
RequiresDaughterBoard	
SpecialRequirements	
RequirementsDescription	
OperatingVoltages	
RemovalConditions	
ManufactureDate	If FRU available See 10.5.2.6
ActiveCooling	
Version	
SerialNumber	If FRU available See 10.5.2.8
PartNumber	If FRU available See 10.5.2.7
Height	
Depth	
Width	
Weight	
IdentifyingDescriptions	
OtherIdentifyingInfo	
UserTracking	
StatusDescriptions	
InstallDate	
Name	If FRU available See 10.5.2.9

Recommended Properties	Notes
Caption	

The following subsections describe procedures for generating values for various properties of CIM classes required in this profile.

10.5.2.1 CIM_Card.Tag

An arbitrary string that uniquely identifies the Physical Element and serves as the Element ' s key. Formulate as string:

(string)<Entity ID from SDR> || ":" || (string)<Entity Instance from SDR> || ":" ||

|| (string)<Sensor Number from SDR>

10.5.2.2 CIM_Card.CreationClassName

Per the DMTF definition of the class property: The name of the class or the subclass used in the creation of an instance. This will typically be "CIM_Card".

Formulate as a string:

<"CIM_Card" or the name of the subclass created>

10.5.2.3 CIM_Card.HostingBoard

Per the MOF, this is supposed to be a Boolean indicating whether or not the card is a Mother board.

Formulate as string:

if Entity ID = 07h (System board)

CIM_Card.HostingBoard = "System board"

else

CIM_Card.HostingBoard = NULL

10.5.2.4 CIM_Card.PackageType

Values for CIM_Card.PackageType are mapped to values of Entity ID from the SDR.

Formulate as integer:

if Entity ID = 07h (System board)

CIM_Card.PackageType = 1 (Other)

2823 CIM_Card.OtherPackageType = “System board”

2824 elseif EntityID = 0Bh (Add-in card)

2825 CIM_Card.PackageType = 1 (Other)

2826 CIM_Card.OtherPackageType = “Add-in card”

2827 elseif EntityID = 0Ch (Front panel board)

2828 CIM_Card.PackageType = 1 (Other)

2829 CIM_Card.OtherPackageType = “Front panel board”

2830 elseif EntityID = 0Dh (Back panel board)

2831 CIM_Card.PackageType = 1 (Other)

2832 CIM_Card.OtherPackageType = “Back panel board”

2833 elseif EntityID = 0Eh (Power system board)

2834 CIM_Card.PackageType = 1 (Other)

2835 CIM_Card.OtherPackageType = “Power system board”

2836 elseif EntityID = 0Fh (Drive backplane)

2837 CIM_Card.PackageType = 1 (Other)

2838 CIM_Card.OtherPackageType = “Drive backplane”

2839 elseif EntityID = 10h (System internal expansion backplane)

2840 CIM_Card.PackageType = 1 (Other)

2841 CIM_Card.OtherPackageType = “System internal expansion backplane”

2842 elseif EntityID = 11h (Other system board)

2843 CIM_Card.PackageType = 1 (Other)

2844 CIM_Card.OtherPackageType = “Other system board”

2845 elseif EntityID = 12h (Processor board)

2846 CIM_Card.PackageType = 1 (Other)

2847 CIM_Card.OtherPackageType = “Processor board”

2848 elseif EntityID = 15h (Power management/power distribution board)

CIM_Card.PackageType = 1 (Other)

CIM_Card.OtherPackageType = “Power management/power distribution board”

elseif EntityID = 16h (Chassis back panel board)

CIM_Card.PackageType = 1 (Other)

CIM_Card.OtherPackageType = “Chassis back panel board”

elseif EntityID = 19h (Other chassis board)

CIM_Card.PackageType = 1 (Other)

CIM_Card.OtherPackageType = “Other chassis board”

elseif EntityID = 26h (Remote management card)

CIM_Card.PackageType = 1 (Other)

CIM_Card.OtherPackageType = “Remote management card”

10.5.2.5 CIM_Card.Manufacturer

Formulate as a string:

(string) associated FRU 1.1 Board Info Area – Board Manufacturer Name field

10.5.2.6 CIM_ Card.ManufacturerDate

Formulate as a string:

(CIM datetime data type) associated FRU 1.1 Board Info Area – Mfg. Date/Time field

10.5.2.7 CIM_ Card.PartNumber

Formulate as a string:

(string) associated FRU 1.1 Board Info Area – Board Part/Model Number field

10.5.2.8 CIM_ Card.SerialNumber

Formulate as a string:

(string) associated FRU 1.1 Board Info Area – Board Serial Number field

10.5.2.9 CIM_ Card.Name

Formulate as a string:

(string) associated FRU 1.1 Board Info Area – Board Product Name field

10.5.2.10 CIM_Card.HealthState

10.5.2.11 CIM_Card.OperationalStatus

10.5.2.12 CIM_Card.Description

Formulate as string:

(string)< CIM_Card.PackageType or CIM_Card.OtherPackageType > || "(" ||
(string)(SensorNumber) || "." || (string)(OwnerLUN) || "." || string)(OwnerID) || "): "
||(string)(EStr(<CIM_Sensor.SensorType>)) || " for " || (string)(<IPMI_SDR_EntityID>)
|| " " || (string)(<IPMI_SDR_EntityInstanceNumber>)

10.5.2.13 CIM_Card.ElementName

Formulate as string:

(string)<)< CIM_Card.PackageType or CIM_Card.OtherPackageType > || "(" ||
(string)<Entity ID from SDR> || ":" || (string)<Entity Instance from SDR> || ")"

10.6 CIM_Chip Mapping

10.6.1 CIM_Chip Methods

None.

10.6.2 CIM_Chip Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that are are listed in the Required section in the following table MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

2897 **Table 20 CIM_Chip Class Properties**

Required Properties	Notes
Tag	Key
CreationClassName	Key
HealthState	
OperationalStatus	
Description	
ElementName	

2898

Recommended Properties	Notes
Manufacturer	
Model	
Version	
VendorEquipmentType	
PoweredOn	
CanBeFRUed	
SKU	
RemovalConditions	
ManufactureDate	
Removable	
Replaceable	
HotSwappable	
UserTracking	
Version	
SerialNumber	
PartNumber	
FormFactor	
StatusDescriptions	
InstallDate	
Name	
Caption	

2899

2900 The following subsections describe procedures for generating values for various properties of
2901 CIM classes required in this profile.

2902 ### 10.6.2.1 CIM_Chip.Tag

2903 An arbitrary string that uniquely identifies the Physical Element and serves as the Element ' s
2904 key. Formulate as string:

2905 (string)<Entity ID from SDR> || ":" || (string)<Entity Instance from SDR> || ":" ||

|| (string)<Sensor Number from SDR>

10.6.2.2 CIM_Chip.CreationClassName

Per the DMTF definition of the class property: The name of the class or the subclass used in the creation of an instance. This will typically be "CIM_Chip".

Formulate as a string:

<"CIM_Chip" or the name of the subclass created>

10.6.2.3 CIM_Chip.HealthState

10.6.2.4 CIM_Chip.OperationalStatus

10.6.2.5 CIM_Chip.Description

Formulate as string:

(string)< SDR entry bytes 49+, see IPMI v1.5 Table 37-1> || "(" ||
(string)(SensorNumber) || "." || (string)(OwnerLUN) || "." || string)(OwnerID) || "): "
||(string)(EStr(<CIM_Sensor.SensorType>)) || " for " || (string)(<IPMI_SDR_EntityID>)
|| " " || (string)(<IPMI_SDR_EntityInstanceNumber>)

10.6.2.6 CIM_Chip.ElementName

Formulate as string:

(string)< SDR entry bytes 49+, see IPMI v1.5 Table 37-1> || "(" || (string)<Entity ID from SDR>
|| ":" || (string)<Entity Instance from SDR> || ")"

10.7 CIM_Chassis Mapping

10.7.1 CIM_Chassis Methods

None Defined.

10.7.2 CIM_Chassis Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Several property vales are dependent on the availability of associated FRU information. The Chassis Info Area as defined in Platform Management FRU Information Storage Definition v1.1 is mapped for this class.

2934 Properties that are listed in the Required section in the following table MUST be implemented as
2935 defined in the subsections following this table unless the Notes column for each property
2936 specifies the defining reference.

2937 **Table 21 CIM_Chassis Class Properties**

Required Properties	Notes
Tag	Key
CreationClassName	Key
ChassisPackageType	
HealthState	
OperationalStatus	
Description	
ElementName	

2938

Recommended Properties	Notes
Manufacturer	
Model	
SKU	
VendorEquipmentType	
VendorCompatibilityString(E)	
PoweredOn	
CanBeFRUed	
LockPresent	
NumberOfPowerCords	
CurrentRequiredOrProduced	
HeatGeneration	
ChassisTypeDescription	
MultipleSystemSupport	
RackMountable	
CableManagementStrategy	
ServicePhilosophy	
ServiceDescriptions	
AudibleAlarm	
VisibleAlarm	
SecurityBreach	
BreachDescription	
IsLocked	
OtherPackageType	
RemovalConditions	
ManufactureDate	
ActiveCooling	

Recommended Properties	Notes
Version	
SerialNumber	If FRU available See Section 10.7.2.5
PartNumber	If FRU available See Section 10.7.2.4
Height	
Depth	
Width	
Weight	
IdentifyingDescriptions	
OtherIdentifyingInfo	
UserTracking	
StatusDescriptions	
InstallDate	
Name	
Caption	

10.7.2.1 CIM_Chassis.Tag

An arbitrary string that uniquely identifies the Physical Element and serves as the Element ' s key. Formulate as a string:

(string)<Entity ID from SDR> || ":" || (string)<Entity Instance from SDR> || ":" ||

|| (string)<Sensor Number from SDR>

10.7.2.2 CIM_Chassis.CreationClassName

Per the DMTF definition of the class property: The name of the class or the subclass used in the creation of an instance. This will typically be "CIM_Chassis".

Formulate as a string:

<"CIM_Chassis" or the name of the subclass created>

10.7.2.3 CIM_Chassis.ChassisPackageType

Values for CIM_Chassis.ChassisPackageType are mapped to values of Entity ID from the SDR.

Formulate as integer:

if Entity ID = 17h (System chassis)

CIM_Chassis.ChassisPackageType = 17 (Main System Chassis)

elseif EntityID = 18h (Sub-chassis)

CIM_Chassis.ChassisPackageType = 19 (SubChassis)

10.7.2.4 CIM_ Chassis.PartNumber

Formulate as a string:

(string) associated FRU 1.1 Chassis Info Area – Chassis Part Number field

10.7.2.5 CIM_ Chassis.SerialNumber

Formulate as a string:

(string) associated FRU 1.1 Chassis Info Area – Chassis Serial Number field

10.7.2.6 CIM_Chassis.HealthState

10.7.2.7 CIM_Chassis.OperationalStatus

10.7.2.8 CIM_Chassis.Description

Formulate as string:

(string)< CIM_Chassis.ChassisPackageType > || "(" || (string)(SensorNumber) || "." || (string)(OwnerLUN) || "." || string)(OwnerID) || "): " ||(string)(EStr(<CIM_Sensor.SensorType>)) || " for " || (string)(<IPMI_SDR_EntityID>) || " " || (string)(<IPMI_SDR_EntityInstanceNumber>)

10.7.2.9 CIM_Chassis.ElementName

Formulate as string:

(string)<CIM_Chassis.ChassisPackageType> || "(" || (string)<Entity ID from SDR> || ":" || (string)<Entity Instance from SDR> || ")"

10.8 CIM_Processor Mapping

10.8.1 CIM_Processor Methods

No Mapping Defined.

10.8.2 CIM_Processor Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

2984 Properties that are listed in the Required section in the following table MUST be implemented as
2985 defined in the subsections following this table unless the Notes column for each property
2986 specifies the defining reference.

2987 **Table 22 CIM_Processor Class Properties**

Required Properties	Notes
SystemCreationClassName	Key
SystemName	Key
CreationClassName	Key
DeviceID	Key
UniqueID	
EnabledState	
HealthState	
OperationalStatus	
Description	
ElementName	

2988

Recommended Properties	Notes
Family	
CurrentClockSpeed	
MaxClockSpeed	
ExternalClockSpeed	
Role	
OtherFamilyDescription	
DataWidth	
AddressWidth	
Stepping	
CPUStatus	
OtherEnabledState	
StatusDescriptions	
Name	
Caption	
Description	
UpgradeMethod	
LoadPercentage	
IdentifyingDescriptions	
AdditionalAvailablity	
OtherIdentifyingInfo	
TimeOfLastChange	
InstallDate	

Recommended Properties	Notes
ElementName	

10.8.2.1 CIM_Processor.SystemCreationClassName

The name of the class or the subclass used in the creation of an instance of the containing system. This will typically be "CIM_ComputerSystem".

Formulate as a string:

<"CIM_ComputerSystem" or the name of the subclass created>

10.8.2.2 CIM_Processor.CreationClassName

Per the DMTF definition of the class property: The name of the class or the subclass used in the creation of an instance. This will typically be "CIM_Processor".

Formulate as a string:

<"CIM_Processor" or the name of the subclass created>

10.8.2.3 CIM_Processor.SystemName

The host name of the containing system. Derived from the CIM_ComputerSystem.Name property of the instance representing the containing system.

Formulate as a string:

<CIM_ComputerSystem.Name>

10.8.2.4 CIM_Processor.DeviceID

Formulate as string:

(string)<Entity ID from SDR> || ":" || (string)<Entity Instance from SDR> || ":" ||

|| (string)<Sensor Number from SDR> || ":" || <"Processor">

10.8.2.5 CIM_Processor.UniqueID

10.8.2.6 CIM_Processor.EnabledState

Formulate as string:

If Sensor offset 08h (Processor Disabled) = 1b
<"Disabled">
Else
<"Enabled"

10.8.2.7 CIM_Processor.Description

Formulate as string:

(string) “Processor” || “(” || (string)(SensorNumber) || “.” || (string)(OwnerLUN) || “.” || string)(OwnerID) || “)” || “ for ” || (string)(<IPMI_SDR_EntityID>) || “ “ || (string)(<IPMI_SDR_EntityInstanceNumber>)

10.8.2.8 CIM_Processor.ElementName

Formulate as string:

(string)< SDR entry bytes 49+, see IPMI v1.5 Table 37-1> || “(” || (string)<Entity ID from SDR> || “:” || (string)<Entity Instance from SDR> || “)”

10.8.2.9 CIM_Processor.HealthState

10.8.2.10 CIM_Processor.OperationalStatus

10.9 CIM_Fan Mapping

10.9.1 CIM_Fan Methods

No Mapping Defined.

10.9.2 CIM_Fan Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that are listed in the Required section in the following table MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

3039 **Table 23 CIM_Fan Class Properties**

Required Properties	Notes
SystemCreationClassName	Key
SystemName	Key
CreationClassName	Key
DeviceID	Key
HealthState	
OperationalStatus	
Description	
ElementName	

3040

Recommended Properties	Notes
VariableSpeed	
DesiredSpeed	
OtherIdentifyingInfo	
AdditionalAvailability	
IdentifyingDescriptions	
AdditionalAvailablity	
EnabledState	
OtherEnabledState	
RequestedState	
EnabledDefault	
TimeOfLastStateChange	
Name	
StatusDescriptions	
InstallDate	
ActiveCooling	
Caption	

3041 ### 10.9.2.1 CIM_Fan.SystemCreationClassName

3042 The name of the class or the subclass used in the creation of an instance of the containing
3043 system. This will typically be “CIM_ComputerSystem”.

3044 Formulate as a string:

3045 <”CIM_ComputerSystem” or the name of the subclass created>

3046

3047 ### 10.9.2.2 CIM_Fan.CreationClassName

3048 Per the DMTF definition of the class property: The name of the class or the subclass used in the
3049 creation of an instance. This will typically be “CIM_Fan”.

Formulate as a string:

<”CIM_Fan” or the name of the subclass created>

10.9.2.3 CIM_Fan.SystemName

The host name of the containing system. Derived from the CIM_ComputerSystem.Name property of the instance representing the containing system.

Formulate as a string:

<CIM_ComputerSystem.Name>

10.9.2.4 CIM_Fan.DeviceID

Formulate as string:

(string)<Entity ID from SDR> || “:” || (string)<Entity Instance from SDR> || “:” ||

|| (string)<Sensor Number from SDR> || “:” || <”Fan”>

10.9.2.5 CIM_Fan.Description

Formulate as string:

“Fan” || “(” || (string)(SensorNumber) || “.” || (string)(OwnerLUN) || “.” || string)(OwnerID) || “) ” || “ for ” || (string)(<IPMI_SDR_EntityID>) || “ “ || (string)(<IPMI_SDR_EntityInstanceNumber>)

10.9.2.6 CIM_Fan.ElementName

Formulate as string:

(string)< SDR entry bytes 49+, see IPMI v1.5 Table 37-1> || “(” || (string)<Entity ID from SDR> || “:” || (string)<Entity Instance from SDR> || “)”

10.9.2.7 CIM_Fan.HealthState

10.9.2.8 CIM_Fan.OperationalStatus

10.10 CIM_PowerSupply Mapping

10.10.1 CIM_PowerSupply Methods

No Mapping Defined.

10.10.2 CIM_PowerSupply Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties are that listed in the Required section in the following table below MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

Table 24 CIM_PowerSupply Class Properties

Required Properties	Notes
SystemCreationClassName	Key
SystemName	Key
CreationClassName	Key
DeviceID	Key
HealthState	
OperationalStatus	
Description	
ElementName	

Recommended Properties	Notes

10.10.2.1 CIM_PowerSupply.SystemCreationClassName

The name of the class or the subclass used in the creation of an instance of the containing system. This will typically be "CIM_ComputerSystem".

Formulate as a string:

10.10.2.2 <"CIM_ComputerSystem" or the name of the subclass created> CIM_PowerSupply.CreationClassName

Per the DMTF definition of the class property: The name of the class or the subclass used in the creation of an instance. This will typically be "CIM_PowerSupply".

Formulate as a string:

<"CIM_PowerSupply" or the name of the subclass created>

10.10.2.3 CIM_PowerSupply.SystemName

The host name of the containing system. Derived from the CIM_ComputerSystem.Name property of the instance representing the containing system.

Formulate as a string:

<CIM_ComputerSystem.Name>

10.10.2.4 CIM_PowerSupply.DeviceID

Formulate as string:

(string)<Entity ID from SDR> || ":" || (string)<Entity Instance from SDR> || ":" ||

|| (string)<Sensor Number from SDR> || ":" || <"Power Supply">

10.10.2.5 CIM_PowerSupply.Description

Formulate as string:

"Power Supply" || "(" || (string)(SensorNumber) || "." || (string)(OwnerLUN) || "." || string)(OwnerID) || ") " || " for " || (string)(<IPMI_SDR_EntityID>) || " " || (string)(<IPMI_SDR_EntityInstanceNumber>)

10.10.2.6 CIM_PowerSupply.ElementName

Formulate as string:

(string)< SDR entry bytes 49+, see IPMI v1.5 Table 37-1> || "(" || (string)<Entity ID from SDR> || ":" || (string)<Entity Instance from SDR> || ")"

10.10.2.7 CIM_PowerSupply.HealthState

10.10.2.8 CIM_PowerSupply.OperationalStatus

10.11 CIM_Memory Mapping

10.11.1 CIM_Memory Methods

No Mapping Defined.

10.11.2 CIM_Memory Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that are listed in the Required section in the following table MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

3125 **Table 25 CIM_Memory Class Properties**

Required Properties	Notes
SystemCreationClassName	Key
SystemName	Key
CreationClassName	Key
DeviceID	Key
HealthState	
OperationalStatus	
Description	
ElementName	

3126

Recommended Properties	Notes
Blocksize	
NumberOfBlocks	
Name	
NameFormat	
NameNamespace	
IdentifyingDescriptions	
EnabledState	
RequestedState	
EnabledDefault	
HealthState	
OperationalStatus	
StartingAddress	
EndingAddress	
Purpose	
Volatile	
ErrorMethodology	
ConsumableBlocks	
IsBasedOnUnderlyingRedundancy	
ExtentStatus	
NoSinglePointOfFailure	
DataRedundancy	

Recommended Properties	Notes
PackageRedundancy	
DeltaReservation	
Primordial	
OtherNameNamespace	
OtherNameFormat	
OtherIdentifyingInfo	
OtherEnabledState	
TimeOfLastStateChange	
InstallDate	
StatusDescriptions	
AdditionalAvailability	
Caption	

10.11.2.1 CIM_Memory.SystemCreationClassName

The name of the class or the subclass used in the creation of an instance of the containing system. This will typically be "CIM_ComputerSystem".

Formulate as a string:

<"CIM_ComputerSystem" or the name of the subclass created>

10.11.2.2 CIM_Memory.CreationClassName

Per the DMTF definition of the class property: The name of the class or the subclass used in the creation of an instance. This will typically be "CIM_Memory".

Formulate as a string:

<"CIM_Memory" or the name of the subclass created>

10.11.2.3 CIM_Memory.SystemName

The host name of the containing system. Derived from the CIM_ComputerSystem.Name property of the instance representing the containing system.

Formulate as a string:

<CIM_ComputerSystem.Name>

10.11.2.4 CIM_Memory.DeviceID

Formulate as string:

(string)<Entity ID from SDR> || ":" || (string)<Entity Instance from SDR> || ":" ||

|| (string)<Sensor Number from SDR> || ":" || <"Memory">

10.11.2.5 CIM_Memory.Description

Formulate as string:

"Memory" || "(" || (string)(SensorNumber) || "." || (string)(OwnerLUN) || "." || string)(OwnerID) || ") " || " for " || (string)(<IPMI_SDR_EntityID>) || " " || (string)(<IPMI_SDR_EntityInstanceNumber>)

10.11.2.6 CIM_Memory.ElementName

Formulate as string:

(string)< SDR entry bytes 49+, see IPMI v1.5 Table 37-1> || "(" || (string)<Entity ID from SDR> || ":" || (string)<Entity Instance from SDR> || ")"

10.11.2.7 CIM_Memory.HealthState

10.11.2.8 CIM_Memory.OperationalStatus

10.12 CIM_Battery Mapping

10.12.1 CIM_Battery Methods

No Mapping Defined.

10.12.2 CIM_Battery Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that are listed in the Required section in the following table MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

3168 **Table 26 CIM_Battery Class Properties**

Required Properties	Notes
SystemCreationClassName	Key
SystemName	Key
CreationClassName	Key
DeviceID	Key
BatteryStatus	
HealthState	
OperationalStatus	
Description	
ElementName	

3169

Recommended Properties	Notes
EnabledState	
RequestedState	
EnabledDefault	
Name	
Description	
EstimatedRunTime	
EstimatedChargeRemaining	
RechargeCount	
MaxRechargeCount	
MaximumError	
RemainingCapacity	
TimeOnBattery	
Chemistry	
DesignCapacty	
FullChargeCapacity	
DesignVoltage	
SmartBatteryVersion	
TimeToFullCharge	
ExpectedLife	
MaxRechargeTime	
TimeOfLastChange	
StatusDescriptions	
InstallDate	
Caption	
ElementName	
IdentifyingDescriptions	
AdditionalAvailablity	
OtherIdentifyingInfo	

Recommended Properties	Notes
OtherEnabledState	

3170 **10.12.2.1 CIM_Battery.SystemCreationClassName**

3171 The name of the class or the subclass used in the creation of an instance of the containing
3172 system. This will typically be "CIM_ComputerSystem".

3173 Formulate as a string:

3174 <"CIM_ComputerSystem" or the name of the subclass created>

3175

3176 **10.12.2.2 CIM_Battery.CreationClassName**

3177 Per the DMTF definition of the class property: The name of the class or the subclass used in the
3178 creation of an instance. This will typically be "CIM_Battery".

3179 Formulate as a string:

3180 <"CIM_Battery" or the name of the subclass created>

3181 **10.12.2.3 CIM_Battery.SystemName**

3182 The host name of the containing system. Derived from the CIM_ComputerSystem.Name
3183 property of the instance representing the containing system.

3184 Formulate as a string:

3185 <CIM_ComputerSystem.Name>

3186 **10.12.2.4 CIM_Battery.DeviceID**

3187 Formulate as string:

3188 (string)<Entity ID from SDR> || ":" || (string)<Entity Instance from SDR> || ":" ||

3189 || (string)<Sensor Number from SDR> || ":" || <"Battery">

3190 **10.12.2.5 CIM_Battery.BatteryStatus**

3191 **10.12.2.6 CIM_Battery.Description**

3192 Formulate as string:

3193 "Battery" || "(" || (string)(SensorNumber) || "." || (string)(OwnerLUN) || "." ||
3194 string)(OwnerID) || ") " || " for " || (string)(<IPMI_SDR_EntityID>) || " " ||
3195 (string)(<IPMI_SDR_EntityInstanceNumber>)

3196

3197 #### 10.12.2.7 CIM_Battery.ElementName

3198 Formulate as string:

3199 (string)< SDR entry bytes 49+, see IPMI v1.5 Table 37-1> || "(" || (string)<Entity ID
3200 from SDR> || ":" || (string)<Entity Instance from SDR> || ")"
3201

3202 #### 10.12.2.8 CIM_Battery.HealthState

3203 #### 10.12.2.9 CIM_Battery.OperationalStatus

3204 ## 10.13 CIM_DiskDrive Mapping

3205 ### 10.13.1 CIM_DiskDrive Methods

3206 No Mapping Defined.

3207 ### 10.13.2 CIM_DiskDrive Properties

3208 REQUIRED properties are properties that MUST be present. RECOMMENDED properties are
3209 properties that MAY be present. For example, there may be properties that are only filled out in
3210 special configurations of the CIM Schema.

3211 Properties that are listed in the Required section in the table below MUST be implemented as
3212 defined in the subsections following this table unless the Notes column for each property
3213 specifies the defining reference.

3214 **Table 27 CIM_DiskDrive Class Properties**

Required Properties	Notes
SystemCreationClassName	Key
SystemName	Key
CreationClassName	Key
DeviceID	Key
HealthState	
OperationalStatus	
Description	
ElementName	

3215

Recommended Properties	Notes
Capabilities	
CapabilityDescriptions	
ErrorMethodology	

Recommended Properties	Notes
CompressionMethod	
NumberOfMediaSupported	
MaxMediaSize	
DefaultBlockSize	
MaxBlockSize	
MinBlockSize	
NeedsCleaning	
MediaIsLocked	
Security	
LastCleaned	
MaxAccessTime	
UncompressedDataRate	
LoadTime	
UnloadTime	
MountCount	
TimeOfLastMount	
TotalMountTime	
UnitsDescription	
MaxUnitsBeforeCleaning	
UnitsUsed	
EnabledState	
RequestedState	
EnabledDefault	
TimeOfLastStateChange	
Name	
Description	
InstallDate	
Caption	
IdentifyingDescriptions	
AdditionalAvailablity	
OtherIdentifyingInfo	
OtherEnabledState	

3216 #### 10.13.2.1 CIM_DiskDrive.SystemCreationClassName

3217 The name of the class or the subclass used in the creation of an instance of the containing
3218 system. This will typically be “CIM_ComputerSystem”.

3219 Formulate as a string:

3220 <”CIM_ComputerSystem” or the name of the subclass created>

3221

3222 #### 10.13.2.2 CIM_DiskDrive.CreationClassName

3223 Per the DMTF definition of the class property: The name of the class or the subclass used in the
3224 creation of an instance. This will typically be “CIM_DiskDrive”.

3225 Formulate as a string:

3226 <”CIM_DiskDrive” or the name of the subclass created>

3227 #### 10.13.2.3 CIM_DiskDrive.SystemName

3228 The host name of the containing system. Derived from the CIM_ComputerSystem.Name
3229 property of the instance representing the containing system.

3230 Formulate as a string:

3231 <CIM_ComputerSystem.Name>

3232 #### 10.13.2.4 CIM_DiskDrive.DeviceID

3233 Formulate as string:

3234 (string)<Entity ID from SDR> || “:” || (string)<Entity Instance from SDR> || “:” ||

3235 || (string)<Sensor Number from SDR> || “:” || <”DiskDrive”>

3236 #### 10.13.2.5 CIM_DiskDrive.Description

3237 Formulate as string:

3238 “DiskDrive” || “(” || (string)(SensorNumber) || “.” || (string)(OwnerLUN) || “.” ||
3239 string)(OwnerID) || “) ” || “ for ” || (string)(<IPMI_SDR_EntityID>) || “ “ ||
3240 (string)(<IPMI_SDR_EntityInstanceNumber>)

3241 #### 10.13.2.6 CIM_DiskDrive.ElementName

3242 Formulate as string:

3243 (string)< SDR entry bytes 49+, see IPMI v1.5 Table 37-1> || “(” || (string)<Entity ID from SDR>
3244 || “:” || (string)<Entity Instance from SDR> || “)”

3245 #### 10.13.2.7 CIM_DiskDrive.HealthState

3246 #### 10.13.2.8 CIM_DiskDrive.OperationalStatus

3247

11 IPMI Watchdog Mapping

11.1 Introduction

The IPMI Watchdog Mapping extends the management capability of the IPMI to CIM mapping by adding the capability to manage the IPMI Watchdog

The following class diagram describes the CIM classes involved in representing the IPMI watchdog.

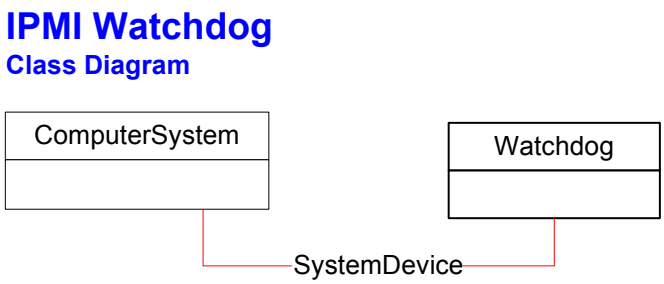


Figure 18 Class Diagram – IPMI Watchdog

11.2 Instance Diagrams

The following instance diagram describes an instance of CIM_Watchdog associated with the instance of CIM_ComputerSystem representing the BMC.

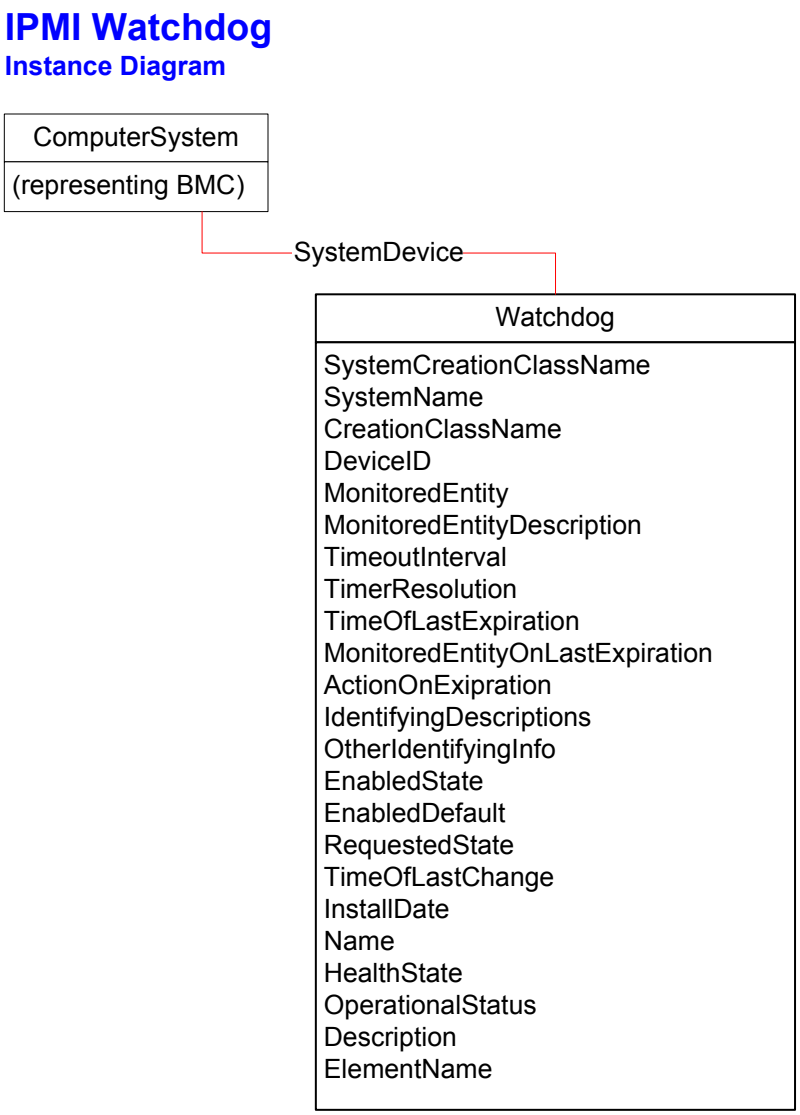


Figure 19 Instance Diagram – IPMI Watchdog

11.3 Mapping Requirements

11.4 CIM_Watchdog Mapping

11.4.1 CIM_Watchdog Methods

11.4.1.1 Method: Reset()

```
uint32 Reset()
```

Per the DMTF definition: The return value should be 0 if the request was successfully executed, 1 if the request is not supported and some other value if an error occurred. This method should map to the IPMI Reset Watchdog Timer command.

A return value of **4 – Failed** would indicate a Reset Watchdog Timer Command response data completion code of 80h meaning "Attempt to start an uninitialized watchdog" as indicated in IPMI v1.5 specification table 21-2.

Return value:
0 – Completed with no error
1 – Not Supported
2 – Unspecified Error
3 – Timeout
4 – Failed
5 – Invalid Parameter
6..0x0FFF – DMTF_Reserved
0x1000..0x7FFF – Method_Reserved
0x8000.. – Vendor_Reserved

11.4.1.2 Method: RequestStateChange

```
uint32 RequestStateChange (
   [IN] unit16 RequestedState,
   [IN] datetime TimeoutPeriod)
```

TimeoutPeriod maps to the Initial Countdown Value bytes for the Set Watchdog Timer command. RequestedState is not used; state changes can be performed by writing appropriate properties in the CIM_Watchdog class instance.

Return value:

0 – Completed with no error
1 – Not Supported
2 – Unspecified Error
3 – Timeout
4 – Failed
5 – Invalid Parameter
6..0x0FFF – DMTF_Reserved
0x1000..0x7FFF – Method_Reserved
0x8000.. – Vendor_Reserved

11.4.1.3 Method: KeepAlive

```
uint32 KeepAlive()
```

A method to re-arm the timer. This method is only used if the MonitoredEntity is "Application". It returns 0 if successful, 1 if unsupported, and any other value if an error occurred. In an IPMI CIM mapping this will always return **1 – Not Supported.**

Return value:

0 – Completed with no error
1 – Not Supported

11.4.2 CIM_Watchdog Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that are listed in the Required section in Table 6 below MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

3320 **Table 28 CIM_Watchdog Class Properties**

Required Properties	Notes
SystemCreationClassName	**Key**
SystemName	**Key**
CreationClassName	**Key**
DeviceID	**Key**
MonitoredEntity	
MonitoredEntityDescription	
TimeoutInterval	
TimerResolution	
TimeOfLastExpiration	
MonitoredEntityOnLastExpiration	
ActionOnExpiration	
EnabledState	
DefaultState	
RequestedState	
Name	
OperationsStatus	
HealthState	
LogState	Pending CR Approval in the DMTF
ActionOnPretimeoutInterval	Pending CR Approval in the DMTF
PretimeoutInterval	Pending CR Approval in the DMTF

3321

Recommended Properties	Notes
OtherIndentifyingInfo	
IdentifyingDescriptions	
AdditionalAvailality	
OtherEnabledState	
TimeOfLastStateChange	
InstallDate	
Caption	
Description	
ElementName	

3322

3323 The following subsections describe procedures for generating values for various properties of
3324 CIM classes required in this profile.

3325 #### 11.4.2.1 CIM_Watchdog.SystemCreationClassName

3326 The name of the class or the subclass used in the creation of an instance of the containing
3327 system. This will typically be “CIM_ComputerSystem”.

Formulate as a string:

<"CIM_ComputerSystem" or the name of the subclass created>

11.4.2.2 CIM_Watchdog.SystemName

The host name of the containing system. Derived from the CIM_ComputerSystem.Name property of the instance representing the containing system.

Formulate as a string:

<CIM_ComputerSystem.Name>

11.4.2.3 CIM_Watchdog.CreationClassName

Per the DMTF definition of the class property: The name of the class or the subclass used in the creation of an instance. This will typically be "CIM_Watchdog".

Formulate as a string:

<"CIM_WatchDog" or the name of the subclass created>

11.4.2.4 CIM_Watchdog.DeviceID

Formulate as a string:

An address or other identifying information to uniquely name the LogicalDevice.

11.4.2.5 CIM_Watchdog.MonitoredEntity

Values { "Unknown" , "Other" , "Operating System" , "Operating System Boot Process" , "Operating System Shutdown Process" , "Firmware Boot Process" , "BIOS Boot Process" , "Application" , "Service Processor" }

ValueMap { "0" , "1" , "2" , "3" , "4" , "5" , "6" , "7" , "8" }

The entity that is currently being monitored by the WatchDog. This property is used to identify the module that is responsible for or whose information is used to re-arm the watchdog at periodic intervals.

IPMI supported entities map to MonitoredEntity values as follows:

001b BIOS FRB2	"6" "BIOS Boot Process"
010b BIOS POST	"5" "Firmware Boot Process"
011b OS Load	"3" "Operating System Boot Process"
100b SMS/OS	"2" "Operating System"
101b OEM	"1" "Other"

MonitoredEntity values not listed above are not supported in this mapping.

11.4.2.6 CIM_Watchdog.MonitoredEntityDecription

Formulate as string:

A string describing more textual information about the monitored entity:

“BIOS FRB2”, “BIOS POST”, “OS Load”, “SMS/OS”, “OEM”

11.4.2.7 CIM_Watchdog.TimeoutInterval

Units ("MicroSeconds")

The timeout interval used by the watchdog, in MicroSeconds.

11.4.2.8 CIM_Watchdog.TimerResolution

Units ("MicroSeconds")

Resolution of the timer. For example, if this value is 100, then the timer can expire anytime between (TimeoutInterval- 100) microseconds or (TimeoutInterval+100) microseconds.

IPMI v1.5 Watchdog resolution is one count/100 milliseconds, or 100000 microseconds. The TimerResolution value will always map to 100000.

11.4.2.9 CIM_Watchdog.TimeOfLastExpiration

Datetime

The time of the last timer expiry.

11.4.2.10 CIM_Watchdog.MonitoredEntityOnLastExpiration

Values { "Unknown" , "Other" , "Operating System" , "Operating System Boot Process" , "Operating System Shutdown Process" , "Firmware Boot Process" , "BIOS Boot Process" , "Application" , "Service Processor" }

ValueMap { "0" , "1" , "2" , "3" , "4" , "5" , "6" , "7" , "8" }

Monitored entity at the time of last timer expiry.

IPMI supported entities map to MonitoredEntityOnLastExpiration values as follows:

001b BIOS FRB2	“6” "BIOS Boot Process"
010b BIOS POST	“5” “Firmware Boot Process”

011b OS Load	“3” “Operating System Boot Process”
100b SMS/OS	“2” “Operating System”
101b OEM	“1” “Other”

MonitoredEntityOnLastExpiration values not listed above are not supported in this mapping.

11.4.2.11 CIM_Watchdog.ActionOnExpiration

Values { "None - Status Only" , "System Reset" , "System Power Off" , "System Power Off, then On" , "Generate System NonMaskableInterrupt (NMI)" , "Generate System Management Interrupt (SMI)" }

ValueMap { "0" , "1" , "2" , "3" , "4" , "5" }

The action that should happen upon the expiry of the watchdog. IPMI supports timeout actions as defined for bits [2:0] of the Set Watchdog Timer/Get Watchdog Timer commands’ Timer Action byte. These map directly to ActionOnExpiration values as follows:

000b No Action	“0” “None – Status Only”
001b Hard Reset	“1” “ System Reset”
010b Power Down	“2” “System Power Off”
011b Power Cycle	“3” “System Power Off, then On”

Values “4” and “5” are not supported in this mapping.

11.4.2.12 CIM_Watchdog.EnabledState

Values { "Unknown" , "Other" , "Enabled" , "Disabled" , "Shutting Down" , "Not Applicable" , "Enabled but Offline" , "In Test" , "Deferred" , "Quiesce" , "Starting" , "DMTF Reserved" , "Vendor Reserved" }

ModelCorrespondence { "CIM_EnabledLogicalElement.OtherEnabledState" }

ValueMap { "0" , "1" , "2" , "3" , "4" , "5" , "6" , "7" , "8" , "9" , "10" , "11..32767" , "32768..65535" }

EnabledState = 5 - Not Applicable (5) indicates the element doesn't support being enabled/disabled.

11.4.2.13 CIM_Watchdog.DefaultState

Formulate as string:

“”

11.4.2.14 CIM_Watchdog.RequestedState

Values { "Enabled" , "Disabled" , "Shut Down" , "No Change" , "Offline" , "Test" , "Deferred" , "Quiesce" , "Reboot" , "Reset" , "Not Applicable" , "DMTF Reserved" , "Vendor Reserved" }

ModelCorrespondence { "CIM_EnabledLogicalElement.EnabledState" }

ValueMap { "2" , "3" , "4" , "5" , "6" , "7" , "8" , "9" , "10" , "11" , "12" , ".." , "32768..65535" }

RequestedState = 12 - Not supported

11.4.2.15 CIM_Watchdog.Name

Per the DMTF definition, the Name property defines the label by which the object is known. When subclassed, the Name property can be overridden to be a Key property.

Formulate as string:

"IPMI_WatchDog"

11.4.2.16 CIM_Watchdog.OperationalStatus

Values { "Unknown" , "Other" , "OK" , "Degraded" , "Stressed" , "Predictive Failure" , "Error" , "Non-Recoverable Error" , "Starting" , "Stopping" , "Stopped" , "In Service" , "No Contact" , "Lost Communication" , "Aborted" , "Dormant" , "Supporting Entity in Error" , "Completed" , "Power Mode" , "DMTF Reserved" , "Vendor Reserved" }
ModelCorrespondence { "CIM_ManagedSystemElement.StatusDescriptions" }
ValueMap { "0" , "1" , "2" , "3" , "4" , "5" , "6" , "7" , "8" , "9" , "10" , "11" , "12" , "13" , "14" , "15" , "16" , "17" , "18" , ".." , "0x8000.." }
ArrayType ("Indexed")

Only the first element of OperationalStatus array is used in this mapping. OperationalStatus Values map to return states from the IPMI v1.5 Get Watchdog Timer command as follows:

Timer Use bit [6] = 1b "timer is started (running)"	"2", "OK"
Timer Use bit [6] = 0b "timer is stopped"	"10", "Stopped"
Error completion code from Get Watchdog Timer command	"6", "Error"
No response from any Get Watchdog Timer command	"12", "No Contact"
No response from most recent Get Watchdog Timer command	"13", "Lost Communication"

11.4.2.17 CIM_Watchdog.HealthState

Values { "Unknown" , "OK" , "Degraded/Warning" , "Minor failure" , "Major failure" , "Critical

failure" , "Non-recoverable error" , "DMTF Reserved" }
ValueMap { "0" , "5" , "10" , "15" , "20" , "25" , "30" , ".." }

“0” – Not supported

11.4.2.18 CIM_Watchdog.LogState

Values { "Unknown" , "Log" , "Do Not Log”}
ValueMap { "0" , "1" , "2" }

Maps to Timer Use bit [7] indicating logging status for the watchdog

11.4.2.19 CIM_Watchdog.ActionOnPretimeoutInterval

Values { "None - Status Only" , "Generate System Management Interrupt (SMI)" , "Generate System NonMaskableInterrupt (NMI)" , "Messaging Interrupt" }

ValueMap { "0" , "1" , "2" , "3" }

The action that should happen when the pretimeout interval for the watchdog timer is passed. IPMI supports timeout actions as defined for bits [2:0] of the Set Watchdog Timer/Get Watchdog Timer commands’ Timer Action byte. These map to ActionOnPretimeoutInterval values as follows:

000b No Action	“0” “None – Status Only”
001b SMI	“1”, "Generate System Management Interrupt (SMI)"
010b NMI/Diagnostic Interrupt	“2” “Generate System NonMaskableInterrupt (NMI)”
011b Messaging Interrupt	“3” “Messaging Interrupt”

11.4.2.20 CIM_Watchdog.PretimeoutInterval

Units ("Seconds")

The pretimeout interval used by the IPMI watchdog, in Seconds.

12 IPMI Software Identity Mapping

12.1 Introduction

The IPMI Software Identity Mapping extends the management capability of the IPMI to CIM mapping by adding the capability to represent software and firmware known to the BMC (including the BMC firmware) where information about the firmware is exposed by the BMC.

The following class diagram describes the CIM classes involved in representing the software and firmware.

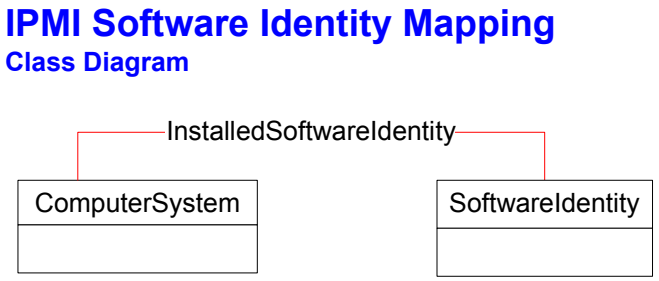


Figure 20 Class Diagram – Software Identity

12.2 Instance Diagrams

The following instance diagram describes an instance of CIM_SoftwareIdentity associated with the instance of CIM_ComputerSystem representing the BMC.

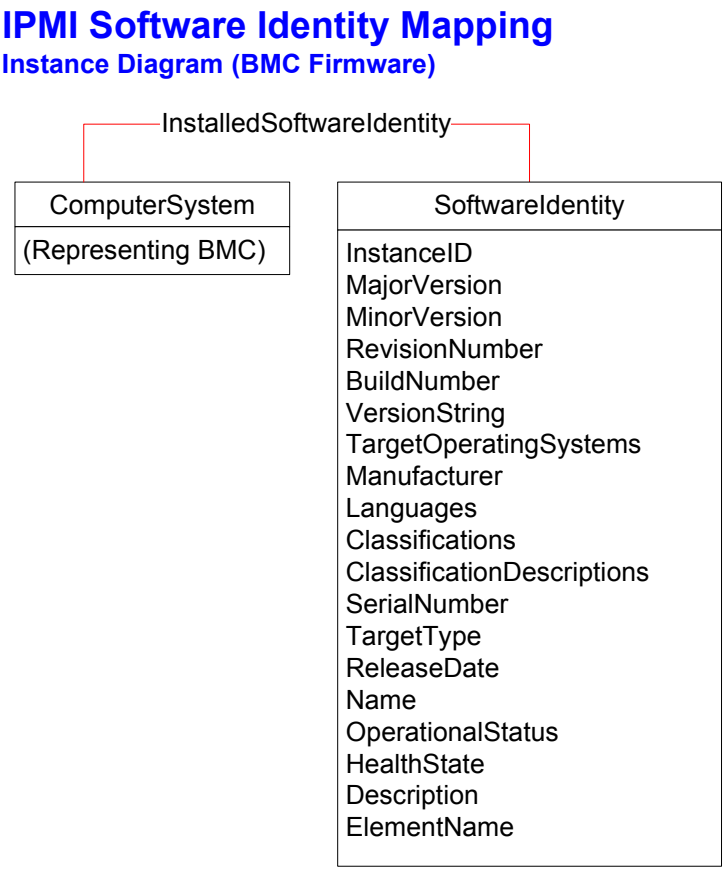


Figure 21 Instance Diagram – Software Identity

12.3 Mapping Requirements

12.4 CIM_SoftwareIdentity Mapping

12.4.1 CIM_SoftwareIdentity Methods

None Defined.

12.4.2 CIM_SoftwareIdentity Properties

REQUIRED properties are properties that MUST be present. RECOMMENDED properties are properties that MAY be present. For example, there may be properties that are only filled out in special configurations of the CIM Schema.

Properties that are listed in the Required section in Table 6 below MUST be implemented as defined in the subsections following this table unless the Notes column for each property specifies the defining reference.

3512 **Table 29 CIM_SoftwareIdentity Class Properties**

Required Properties	Notes
InstanceID	**Key**
RevisionNumber	
VersionString	
Manufacturer	
Name	
OperationalStatus	
HealthState	

3513

Recommended Properties	Notes
TargetOperatingSystems	
ReleaseDate	
BuildNumber	
MajorVersion	
MinorVersion	
Languages	
Classifications	
ClassificationDescriotions	
SerialNumber	
TargetType	
Caption	
Description	
ElementName	

3514

3515 The following subsections describe procedures for generating values for various properties of
3516 CIM classes required in this profile.

3517 #### 12.4.2.1 CIM_SoftwareIdentity.InstanceID

3518 Per the DMTF definition of the class property: In order to ensure uniqueness within the
3519 NameSpace, the value of InstanceID SHOULD be constructed using the following ’ preferred ’
3520 algorithm: < OrgID > : < LocalID > Where < OrgID > and < LocalID > are separated by a colon
3521 ’ : ’ , and where < OrgID > MUST include a copyrighted, trademarked or otherwise unique name
3522 that is owned by the business entity creating/defining the InstanceID, or is a registered ID that is
3523 assigned to the business entity by a recognized global authority.

3524 Formulate as a string:

3525 <”IPMI:” || DeviceID from the Get Device ID Command>

3526

12.4.2.2 CIM_SoftwareIdentity.RevisionNumber

Per the DMTF definition of the class property: This property is defined as a numeric value to allow the determination of ’ newer ’ vs. ’ older ’ releases. A ’ newer ’ revision is indicated by a larger numeric value.

Formulate as a string:

<Firmware Revision 1 from the Get Device ID Command || “.”

|| Firmware Revision 2 from the Get Device ID Command>

12.4.2.3 CIM_SoftwareIdentity.VersionString

Per the DMTF definition of the class property: A string representing the complete software version information.

Formulate as a string:

<<Firmware Revision 1 from the Get Device ID Command || “.”

|| Firmware Revision 2 from the Get Device ID Command || “.”

|| Auxillary Firmware Revision Information from the Get Device ID Command>

12.4.2.4 CIM_SoftwareIdentity.Manufacturer

Per the DMTF definition of the class property: Manufacturer of this software.

Formulate as a string:

<<Manufacturer ID from the Get Device ID Command>

12.4.2.5 CIM_SoftwareIdentity.Name

Formulate as a string:

< “IPMI BMC Firmware “ ||

|| Firmware Revision 1 from the Get Device ID Command || “.”

|| Firmware Revision 2 from the Get Device ID Command>

12.4.2.6 CIM_SoftwareIdentity.OperationalStatus

12.4.2.7 CIM_SoftwareIdentity.HealthState

13 Acknowledgments

Editor

Jon Hass - Dell Inc. for the IPMI CIM Subcommittee of the IPMI Forum

Contributors

Jon Hass - Dell Inc.

Steffen C. Hulegaard - OSA Inc.

Frank Li – OSA Inc.

Arvind Kumar – Intel Corporation

Tom Slaight - Intel Corporation

Wayne Weilnau – Dell Inc.

Chandra Mugunda – Dell Inc.

Phil Chidester - Dell Inc.

Steve Lyle – Hewlett-Packard Company

Kevin Howard – Hewlett-Packard Company

End of Document